Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Titulació:

Enginyeria Industrial (pla 2003)

Alumne (nom i cognoms):

Marcelo Germán López Pérez

Títol PFC:

Proyecto de diseño de una red inalámbrica de sensores de bajo coste

Director del PFC:

David González Diez

Convocatòria de lliurament del PFC:

2014/15 Q1

Contingut d'aquest volum: **-MEMORIA-**

Índice de Contenido

Índice de figuras

Índice de tablas

1. Definición del proyecto

1.1. Objeto

El presente proyecto consiste en el diseño de una red inalámbrica de sensores fácilmente configurable y adaptable para diferentes aplicaciones. Concretamente este proyecto se centra en el diseño de uno de los nodos sensores, el entrono de configuración de la red y el entorno de configuración y procesado de datos de la red.

1.2. Justificación

Hoy en día, las nuevas tecnologías de la información permiten la comunicación a niveles nunca antes vistos. Los flujos de información son cada vez mayores, tanto cuantitativa como cualitativamente, y con el incremento constante de la capacidad de procesamiento de los ordenadores, nuestra capacidad para administrarla y controlarla, también. Con ello es posible recoger información del mundo que nos rodea, cuyo valor es muy importante no sólo para conocerlo y comprenderlo mejor sino también para actuar de forma más eficiente.

La creciente automatización de nuestro entorno requiere de información fiable para su correcto funcionamiento, y la monitorización de éste permite aumentar la calidad de vida creando un espacio más sano (controlando la calidad de los alimentos que consumimos o la calidad del aire que respiramos) y seguro para vivir (previniendo frente a desastres naturales).

Esta es la misión de las redes de sensores: la recogida de información útil y fiable para incrementar el conocimiento y el control sobre el entorno.

Las redes de sensores están formadas por una gran cantidad de nodos, cada uno con un área de influencia pequeña. De esta manera, el conjunto de nodos forma un sistema cooperativo con influencia sobre una gran área. Este sistema, formado por una miríada de pequeños nodos tiene ventajas frente a otro formado por nodos más grandes y potentes. En primer lugar permiten obtener una información mucho más localizada y precisa del entorno en el cual se encuentran integrados. En segundo lugar, los pequeños nodos de las redes de sensores se encuentran muy próximos al fenómeno en observación, por lo que la relación señal-ruido es mayor permitiendo obtener datos de mejor calidad más fácilmente.

En particular, las redes inalámbricas de sensores o WSN (del inglés Wireless Sensor Network) permiten la monitorización de espacios mucho más extensos y/o extremos gracias a que no necesitan de un enlace físico (cables) para funcionar. Otra ventaja de estos sistemas digitales es la integración con el resto de tecnologías, de tal manera que la información puede transferirse rápidamente a Internet y estar disponible para su acceso y uso en cualquier parte del mundo.

De la misma forma, a través de la tecnología de la WSN, el operador de la red puede actuar fácilmente sobre el sistema, realizando las acciones pertinentes para controlar y corregir su funcionamiento.

En este proyecto se plantea el diseño de una red que posea las características mencionadas y que incluya otras propiedades como bajo coste, reducido consumo energy fácil y rápida implementación.

1.3. Alcance

El alcance de este proyecto puede dividirse en cuatro bloques:

1. Definición de las prestaciones requeridas por el sistema.
2. Definición del hardware necesario y justificación de los componentes seleccionados.
3. Software. Configuración e implementación de los módulos seleccionados.
4. Implementación: ensayos de funcionamiento. Definición, ejecución y análisis de los resultados.

No se tratarán específicamente aspectos concretos de sensores, condicionamiento de señal, interficies ni actuadores.

1.4. Especificaciones

Las especificaciones del diseño de la red son las que siguen:

- Uso de un microcontrolador Arduino en los nodos sensores por su fácil manejo, relación calidad/precio y entorno libre de desarrollo.
- Uso del estándar Zigbee para las comunicaciones mediante módulos Xbee compatibles con el entorno Arduino.
- Implementación del lenguaje Processing para el software de procesado de datos, por su fácil manejo, entorno libre de funcionamiento y alta compatibilidad con Arduino.
- El coste final de cada nodo sensor será inferior a 50 € (sin tener en cuenta sensores, acondicionadores de señal ni actuadores).

2. Antecedentes y estado del arte

2.1. Reseña histórica

Como muchos proyectos tecnológicos, las WSN tienen su origen en aplicaciones militares, muy alejada de las actuales aplicaciones de consumo e industriales. La primera red de sensores de la que se tiene constancia fue desarrollada en la década de 1950 para la US Navy. El Sistema de Vigilancia Sónica o SOSUS (del inglés Sound Survilliance System) consistía en arrays de hidrófonos desplegados por zonas estratégicas de los océanos Atlántico y Pacífico para detectar la presencia de submarinos soviéticos. Estos hidrófonos estaban conectados a estaciones en tierra por cables donde la información era recogida 24 horas al día y analizada por personal experto [1].

Con el desarrollo de la electrónica y el hardware para las comunicaciones durante la década de 1960 y 1970, la DARPA (siglas en inglés de Defense Advance Research Projects Agency) inició el programa de Distribuuted Sensor Networks (DSN) a finales de la década de 1970 y principios de la siguiente.

El desarrollo de este programa dentro del mundo académico, que actuó como partner tecnológico, permitió a las WSN introducirse en el ámbito civil para programas de investigación (monitorización de la calidad del aire o para la prevención de incendios forestales) y en el campo académico en general. Luego se extenderían hacia aplicaciones industriales como redes de suministro eléctrico, tratamiento de aguas o automatización puntual en fábricas.

A pesar de que cada vez hay más mercados para la integración de WSN y la demanda crece año a año, existen todavía algunos impedimentos para que su uso masivo. La falta de estandarización es uno de estos factores: muchos de estos sistemas están fabricados con protocolos propios y resultan caros de implementar. Varias iniciativas y organizaciones tratan de remediar este problema impulsando estándares para promover el uso de las WSN, como la Zigbee Alliance o las normalizaciones del IEEE (Institute of Electrical and Electronic Engineers).

Otros problemas técnicos como el coste del hardware y de implementación y el consumo energético se van resolviendo día a día con el desarrollo de electrónica más eficiente y de baterías mejores y más duraderas.

2.2. Aplicaciones

Una muestra del potencial de las WSN en muchas y muy variadas aplicaciones se puede encontrar en el caso de Waspmote, desarrollada por la empresa Zaragozana Libelium. Se trata de una plataforma para el desarrollo de WSN que incorpora una tecnología de código abierto y en la actualidad dispone de más de 60 sensores para medir gases, químicos, ruido, radiación, presencia y tamaño de objetos, y demás.

Figura 1. Nodo Waspmote. Fuente: Libelium.

Entre los proyectos que actualmente está desarrollando Libelium con Waspmote, se pueden citar los siguientes: [2]

Smart Santander Project: Consiste en el despliegue de una WSN bajo protocolos de comunicación ZigBee para la monitorización de las condiciones ambientales y de plazas de aparcamiento en la ciudad de Santander. Para ello se han desplegado 1125 Waspmotes formando una de las mayores WSN del mundo, controlando parámetros como temperatura, luminosidad, ruido, CO y plazas libres.

Siega System: el objetivo de esta red, desplegada en una viña en Pontevedra para pruebas piloto por el Grupo Austen, es desarrollar modelos de predicción para la aparición de plagas. Esto permitirá anticiparse a su ataque y reducir las pérdidas que las plagas ocasionan en las cosechas. Se desplegaron 10 Waspmote

con sensores para monitorizar el clima y el suelo de la viña y actuadores para controlar la climatización y la irrigación.

PRETESIC: proyecto dedicado a medir la calidad del agua en las cloacas de la ciudad de Valencia. Consiste en el despliegue de nodos Waspmote por una unidad móvil para medir condiciones como temperatura, PH, conductividad o demanda química de oxígeno. La información recogida se envía por internet hacia un servidor donde los datos serán almacenados y estudiados.

SISVIA Vigilancia y seguimiento: el objetivo de este proyecto es la monitorización de los bosques para detectar y controlar incendios forestales. Para esto se desplegó una WSN de 90 Waspmotes en Asturias, que miden cuatro parámetros: temperatura, humedad relativa, CO y CO_2. Si alguno de los parámetros supera un umbral preestablecido, el sistema envía una alarma automáticamente a las brigadas de bomberos, indicando su posición y condiciones a tiempo real.

2.3. Marco tecnológico

2.3.1. Sensores. El estándar IEEE 1451

Lanzado en 1993 por el IEEE, este estándar tiene por objetivo resolver y facilitar la conexión y el uso de sensores en las DSN. Está compuesto por una serie de especificaciones para el desarrollo de los llamados Smart Sensors, sensores y/o actuadores que incluyan cateréticas extra más allá de las necesarias para la detección de los fenómenos, como condicionamiento y procesamiento de señal, capacidad de decisión o funciones de alarma.

Este estándar define un conjunto de interfaces de comunicación para conectar transductores inteligentes (sensores y/o actuadores) a sistemas basados en microprocesadores y proporciona un conjunto de protocolos para redes de comunicación. Para ello se definen unos elementos básicos o módulos necesarios para el correcto funcionamiento de los sensores y la transmisión de la señal:

- NCAP (Network Capable Application Processor): es el bloque constructivo que incluye los algoritmos propios de la aplicación para la que utilicen los sensores y funciones de comunicación en red.
- STIM (Smart Transducer Interface Module): bloque que incluye las unidades de sensores y actuadores, transductores y elementos de condicionamiento de señal. Para su correcta utilización es necesaria la incorporación de los TEDS.

- TEDS (Transducer Electronic Data sheet): son una serie de datos propios de cada sensor conectado al STIM que se utilizan para la autoidentificación y el correcto funcionamiento los primeros. En estas "fichas" almacenadas en los sensores se incluye información como identificación del transductor, rango de medición o datos de calibración.

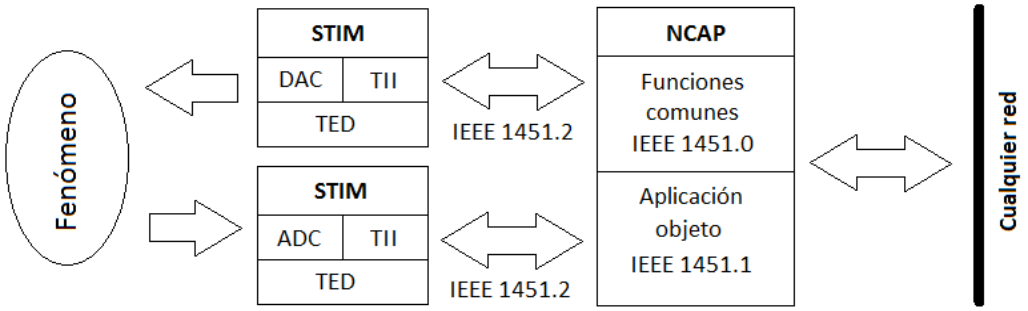

Figura 2. Diagrama de bloques del IEEE 1451 para smart transducer interface. Fuente: propia

A pesar de que el estándar tiene más de 10 años, su adopción es actualmente limitada debido a la complejidad de algunas de sus partes, como el TII (Transducer Independent Interface) que puede verse en la figura anterior; o la dificultad para preparar los TEDS. Sin embargo, las continuas revisiones permiten mejorar el estándar y su adopción es cada vez mayor.

Aunque este proyecto no se haya desarrollado teniendo en cuenta este estándar, se ha considerado conveniente nombrarlo para que el lector lo conozca. Llegado el caso, la plataforma desarrollada podría adaptarse al IEEE 1451, mejorando así sus prestaciones.

2.3.2. Comunicaciones: IEEE 802.15.4

A medida que avanza la electrónica y aumenta la eficiencia, la comunicación de los nodos en las WSN irá dominando el consumo energético. Por tanto, elegir correctamente la forma de comunicación es fundamental. Dependiendo del alcance que deban tener los nodos y de la velocidad de transmisión que se desee para la red, existen diferentes tecnologías disponibles en el mercado, la mayoría de las cuales se encuentran estandarizadas:

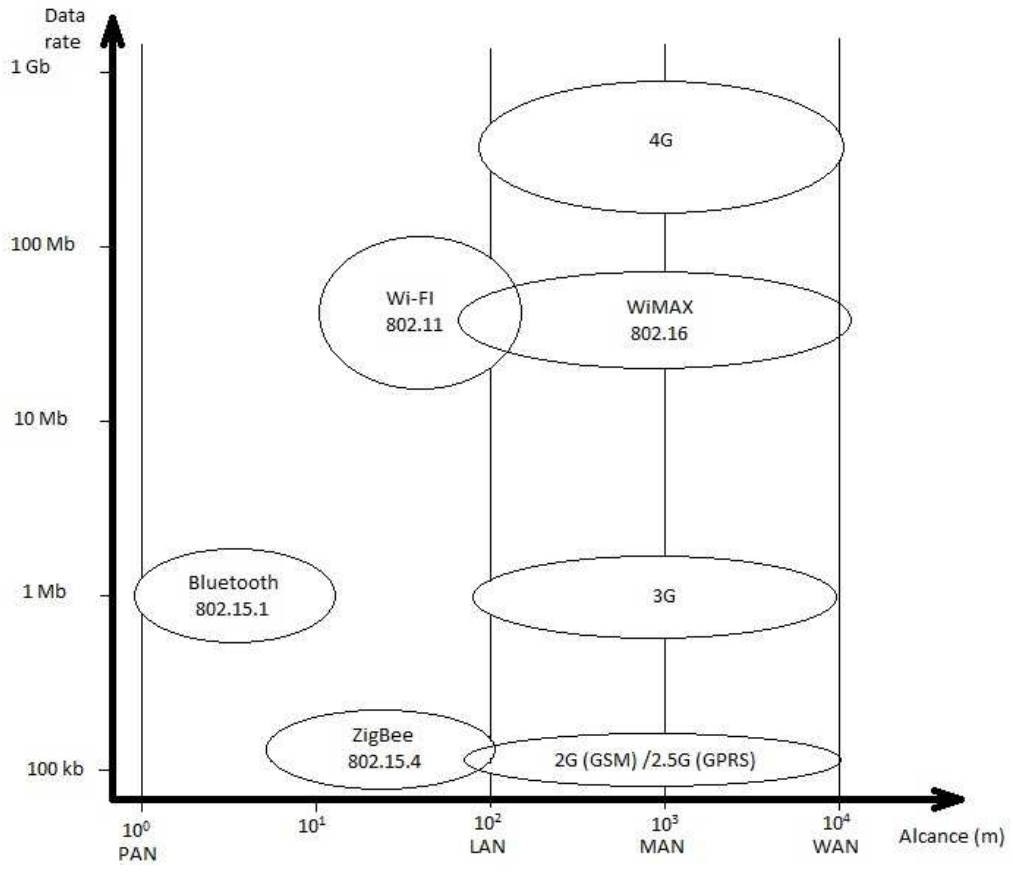

Figura 3. Servicios y área de cobertura de diferentes sistemas inalámbricos. Fuente: propia

Teniendo en cuenta que en este trabajo se desarrolla una WSN para facilitar la incorporación de la tecnología en nuevas aplicaciones industriales y de consumo, el alcance necesario no sería superior a algunas decenas de metros. Por esta razón y para mantener el bajo coste se escogió la especificación ZigBee (como se indicó en la definición del proyecto), desarrollada inicialmente para aplicaciones de demótica y automatización. Otras ventajas del protocolo ZigBee son que se trata de protocolo abierto, con capacidad para operar en productos de diferentes fabricantes, y cuyas actualizaciones se pueden conseguir fácilmente a través de Internet.

El estándar IEEE 802.15.4 es la base sobre la que se desarrolla la especificación ZigBee. Fue desarrollado para responder a la necesidad de una red inalámbrica de área personal con baja tasa de transmisión de datos. En aquellas aplicaciones donde el estándar 802.11 (Wi-fi) resulta excesivo por sus capacidades y consumo energético y el estándar 802.15.1 (Bluetooth) resulta demasiado complejo, el nuevo estándar permite la transmisión de datos con bajo consumo energético y

empleando un hardware más sencillo que las otras, lo que resulta en costes menores. [3]

De acuerdo con el modelo OSI (Open System Interconnection), que es el modelo de referencia que establece un conjunto de reglas aplicables para la comunicación de sistemas informáticos en red, el estándar IEEE 802.15.4 define el nivel físico y el control de acceso al medio (capas más bajas). [4]

Tabla 1. Pila del modelo OSI.

LA PILA OSI
Nivel de aplicación
Nivel de presentación
Nivel de sesión
Nivel de transporte
Nivel de red
Nivel de enlace de datos
Nivel físico

La capa de red es la responsable de la topología de construcción y mantenimiento de la misma, además de nombrarla y de los servicios de enlace que incorpora las tareas necesarias de direccionamiento y seguridad. Las redes formadas bajo el estándar IEEE 802.15.4 se espera que se organicen y se mantengan funcionando automáticamente. Este estándar soporta múltiples topologías de red, entre las que aparecen las tres típicas: estrella, mesh y árbol. La topología escogida dependerá de la aplicación a la que se desee orientar. Por ejemplo, una red pequeña gobernada por un ordenador central utilizará una configuración en estrella de baja potencia, mientras que una aplicación con nodos mucho más espaciados como una fábrica o con un área de cobertura mayor será mejor en una disposición mesh.

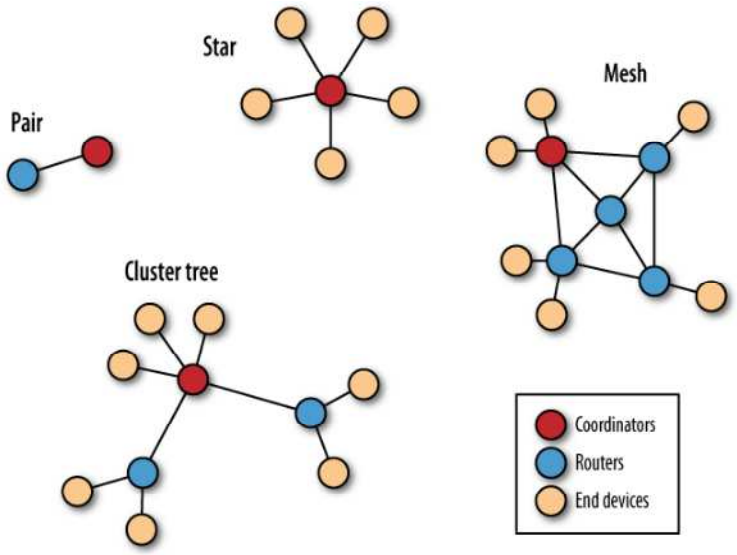

Figura 4. Diferentes topologías de red admitidas por el estándar IEEE 802.15.4. [5]

En cuanto a la capa de enlace de datos, comentar que una parte es compartida con los estándares IEEE 802, el LLC (Link Logical Control, o de control de enlace lógico), mientras que una capa MAC (Medium Access Control, o de control de acceso al medio) simplificada le otorga flexibilidad para adaptarse a múltiples aplicaciones.

El estándar IEEE 802.15.4 utiliza la técnica DSSS (Direct Sequence Spread Spectrum) para la modulación de la señal, lo que permite disminuir la potencia de emisión necesaria y reducir los efectos del ruido y las interferencias.

A nivel físico se ofrecen dos capas cuya principal diferencia es la banda de frecuencia en la que operan. Por un lado, la de 2,4 GHz es ampliamente utilizada en el mundo industrial, médico y científico (de ahí que se conozca también como banda ISM: industrial, scientific and medical) y es una banda disponible sin licencia a nivel mundial. Por otro lado, la banda de 865/915 MHz, disponibles sólo en Europa y Estados Unidos, respectivamente.

Existen ventajas y desventajas para cada una. Mientras la primera es ampliamente utilizada y más fácil y barata de manufacturar, la banda de transmisión de 2,4 GHz se encuentra cada vez más congestionada (Microondas, Wi-fi, Bluetooth) y el riesgo de interferencias es cada vez mayor. Por otro lado, la capa de 865/915 MHz no se encuentra tan extendida y su disponibilidad a nivel internacional no está asegurada, pero la transmisión de datos no se ve tan afectada por interferencias y ofrecen un radio de alcance mayor con menores consumos energéticos.

Otra diferencia entre las dos capas es la tasa máxima de transmisión. Mientras la PHY (capa física) de 2,4 GHz ofrece un máximo de 250 kb/s, la de 865/915 ofrece tasas máximas de 20 y 40 kb/s, para cada frecuencia respectivamente.

2.3.3. ZigBee

En las capas superiores de la pila OSI se encuentra el protocolo ZigBee. Estas capas dotan a los dispositivos de tres rasgos adicionales:

- Enrutado (routing): tablas de enrutado que indican a un dispositivo cómo pasar información a través de una serie de radios hasta su destino.
- Creación de redes Ad-hoc: Este proceso permite crear redes de radios de forma automática sin la intervención de personas.
- Self-healing mesh: es un proceso automático que permite reconfigurar la red en caso de que alguno de los dispositivos falle y mantener así rutas de comunicación.

En una red ZigBee se pueden encontrar tres tipos de dispositvos:

- Coordinador (***coordinator***): toda red ZigBee tendrá un único dispositivo de este tipo. Es el encargado de formar la red, otorgar direcciones a los demás dispositivos, mantener la seguridad, mantenerla sana y administrar otras funciones de definición de la red.
- Enrutador *(**router***): los enrutadores son dispositivos que puede recibir, enviar o pasar información. Su función principal es pasar información entre dispositivos que se encuentran muy alejados para establecer comunicación directa. Son dispositivos que suelen ir conectados a la red eléctrica ya que, por lo general, deben mantenerse siempre encendidos.
- Terminal (***end device***): son versiones simplificadas de un enrutador ya que sólo pueden recibir y enviar información. Al no tener que asumir tantas funciones pueden utilizar un hardware más sencillo y pueden apagarse temporalmente, ahorrando energía. Todo nodo terminal necesita de un router o un coordinador que actúe como su tutor, ayudándole a asociarse a la red y guardando mensajes dirigidos a él cuando se encuentra en estado de ahorro de energía o dormido.

El protocolo ZigBee permite crear redes multi-hop mesh. En esta topología la comunicación se realiza peer-to-peer pasando la información entre nodos hasta su destino. En las redes mesh todos los nodos tiene el mismo papel y sirven tanto como dispositivo final y routers. La formación y reconfiguración automática

de la red y el establecimiento automático de las rutas de comunicación son propiedades de estas redes que aseguran que el mensaje llegue a su destino.

2.3.4. Arduino y ZigBee

Según su sitio web, "Arduino es una plataforma de prototipado electrónico de código abierto, basado en hardware y software fácil de utilizar" [6]. Nacida en 2005, la iniciativa se ha extendido por todo el mundo y se ha vuelto un estandarte de la comunidad DIY (Do It Yourself). Los proyectos desarrollados con esta plataforma son tan variados que sería imposible describirlos aquí. Baste decir que pueden encontrarse desde lámparas controladas por Arduino para modificar la intensidad y el color de la luz, hasta el proyecto ArduSat, lanzado en 2012, que consiste en un satélite (de estándar CubeSat) de código abierto que utiliza la plataforma Arduino como medio para acercar la ciencia espacial al público.

Teniendo en cuenta su amplia utilización, sus enormes prestaciones, desarrolladas por una comunidad que se extiende por todo el mundo y se comunica gracias a Internet, y su magnífica relación calidad-precio, Arduino es la opción, discutiblemente, más lógica en la que basarse para desarrollar la WSN.

Existen actualmente numerosas versiones de placas Arduino, desde las más sencilla y extendida Arduino UNO, con un procesador de 8 bits a 16 MHz y 32 kByte de memoria, hasta la placa compatible Intel Galileo, con un procesador de 32 bits a 400 MHz y hasta 512 kByte dedicados a almacenamiento, aumentables hasta 32 GByte por la ranura de expansión para tarjetas SD. Además de las anteriores, existen versiones Arduinos en diferentes formas y tamaños, como la versión Lylipad, diseñada para poder ser cosida a la ropa, o las versiones mini, micro y nano, de reducido tamaño.

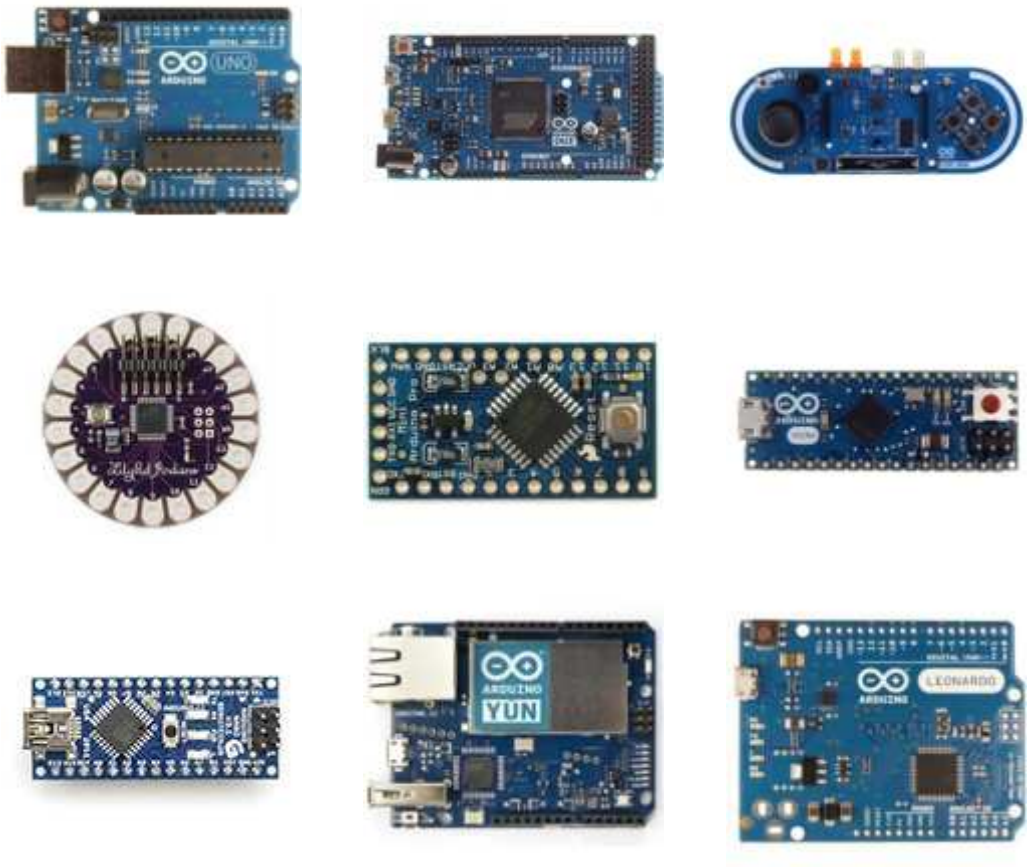

Figura 5. Diferentes modelos de placas Arduino. De izquierda a derecha y de arriba a abajo: UNO, DUE, Esplora, Lylipad, Pro Mini, Micro, Nano, Yun y Leonardo. [6]

La forma más extendida de incorporar una radio con protocolo ZigBee a una placa Arduino son los módulos XBee. Se tratan de una familia de transceptores para comunicación inalámbrica desarrollados por Digi International Inc. Cabe destacar que XBee no es lo mismo que ZigBee, como ZigBee no es lo mismo que IEEE 802.15.4. ZigBee es un protocolo utilizado para comunicación inalámbrica en red basado en el estándar IEEE 802.15.4, e XBee es una marca de transceptores. No todos los modelos de XBee utilizan ZigBee. Por ejemplo, existen los modelos XBee Wi-Fi, o XBee DigiMesh (un protocolo desarrollado por Digi para la comunicación en redes mesh). De la familia XBee, sólo los modelos de la Serie 2 soportan el protocolo ZigBee completo; las radios de la Serie 1 utilizan el protocolo más sencillo basado en el estándar IEEE 802.15.4. Teniendo en cuenta esto, en este proyecto se utilizarán XBee Serie 2 para las comunicaciones inalámbricas de la red.

Las radios XBee son dispositivo muy fáciles de implementar con Arduino ya que se comunican a través del puerto serie de sus respectivos UART. Configuradas en modo transparente/de comandos las radios son totalmente transparentes para

Arduino, transmitiendo los datos como si lo hiciera a través del puerto USB. En este modo la configuración de las radios se puede realizar mediante comandos AT a través de cualquier monitor serie.

Los módulos disponen también de un pequeño procesador que le permite actuar por sí solo. Gracias a él los módulos XBee disponen de entradas y salidas digitales y de un ADC de 10 bits que le permite realizar algunas mediciones y actuación sencilla.

Para aprovechar al máximo las características de XBee, como la creación de redes mesh, el uso de entradas y salidas y una mayor eficiencia en las comunicaciones, lo mejor es implementar el modo de funcionamiento API (application programming interface). Se trata de un protocolo de comunicaciones entre las radios y su *host* (el microcontrolador Arduino o el ordenador) que emplea unos marcos o *frames* para realizar tareas como recepción y transmisión de datos, configuración de la radio, lectura de los pines analógicos y digitales, y demás.

Las radios configuradas en modo permiten direccionar los mensajes (escribiendo la dirección del destinatario en el marco de correspondiente), algo que no es posible en modo transparente, que limita el tamaño y la configuración de la red. En modo API se puede crear una verdadera red mesh, además de que las radios se pueden configurar de forma remota cuando se encuentran en este modo, una característica muy útil cuando las radios funcionan en entornos aislados.

Cabe destacar que en modo API las radios XBee son más fáciles de configurar a través de X-CTU, una aplicación desarrollada por DIGI para tal fin. Esta aplicación se utiliza además para actualizar el firmware de las radios, pudiendo modificar el perfil del dispositivo: coordinador, router o end device. Adimás, la última versión dispone de una consola de comunicaciones que permite construir marcos API de forma sencilla. Por todo ello esta será la forma principal en que se configurarán las radios de la red diseñada.

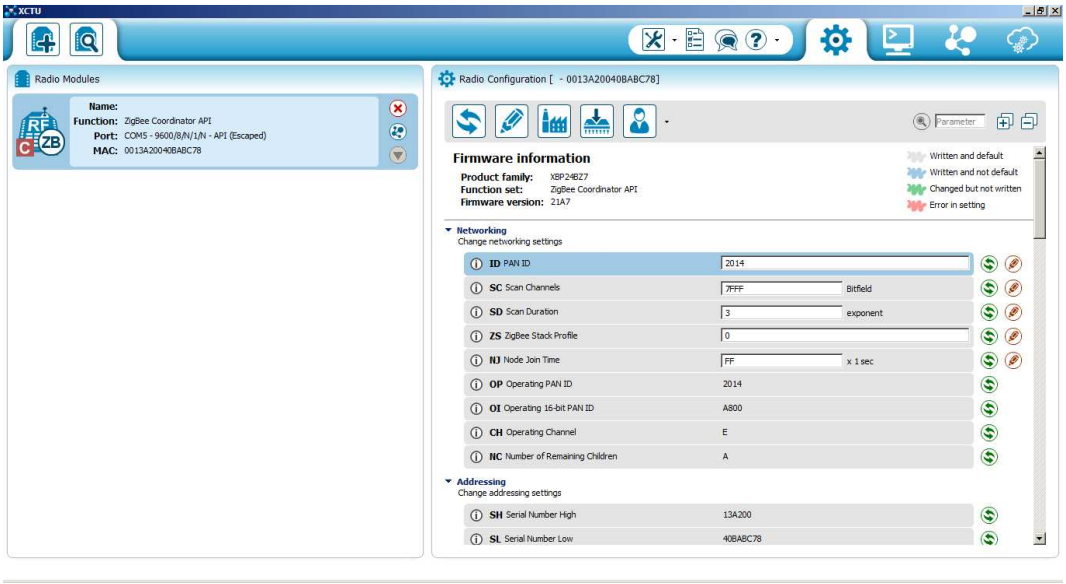

Figura 6. Ventana principal de XBee mostrando la configuración actual de un nodo coordinador en modo API. Fuente: propia.

Conocido el marco tecnológico actual sobre el que se desarrollan las WSN y algunas de las aplicaciones en las que se pueden implementar, en los siguientes capítulos se abordan los aspectos de creación de los nodos y desarrollo del software para la implementación de una WSN.

3.Propiedades y prestaciones de la red

A la hora de diseñar la red: hardware y software, se tuvieron en cuenta una serie de prestaciones y propiedades que el sistema debía de disponer para asegurar su correcto funcionamiento, de acuerdo a las especificaciones establecidas y otras características propias de las WSN.

En todo caso, el proyecto aquí desarrollado intenta servir de base para la implementación de las WSN, y la elección final deberá ajustarse a las necesidades del usuario para disminuir los costes y maximizar las prestaciones.

3.1. Clasificación de los nodos

Los nodos de la red se clasificarán según su papel en la red y siguiendo una estructura similar a la clasificación del protocolo ZigBee:

- Central: nodo principal de la red. Se trata del nodo central en el que se vuelcan los datos recogidos. Este nodo incorporará la radio XBee configurada como coordinador.
- Nodos sensores: son aquellos que se encargan de realizar las mediciones para luego transmitirlas al nodo central. También pueden disponer de capacidad para realizar alguna acción correctora.
- Router: son dispositivos cuya función principal es servir de enlace a las comunicaciones entre los nodos sensores y el nodo coordinador.

Esta clasificación no está necesariamente relacionada con el tipo de módulo XBee que los nodos incorporarán; eso dependerá del modo de funcionamiento de las radios y de la topología de la red.

3.2. Modos de funcionamiento

El funcionamiento de los nodos sensores de una WSN se puede clasificar según el muestreo del fenómeno:

- Cíclico: muestreo periódico del fenómeno en observación. Asociado a registros y control de actividades.
- Activación por excepción: muestreo por estímulo externo del sensor. Asociado a sistemas de aviso y alarmas.

Los nodos sensores también se pueden clasificar de acuerdo a su papel sobre el fenómeno:

- Monitorización: los nodos sólo se encargan de observar mediante muestreo el fenómeno en curso.
- Control: el nodo tiene capacidad añadida de influenciar el fenómeno mediante la aplicación de una acción. Esta acción se puede realizar bien por la transmisión de un comando o por la configuración de parámetros. En este caso los nodos además de los sensores del fenómeno incorporan actuadores y sensores para aplicar la acción correctora.

La red diseñada debe poder configurarse para funcionar en cualquiera de los modos descritos anteriormente.

3.3. Mensaje

La estructura de los mensajes debe ser los más sencilla posible para minimizar el uso de la red y el consumo de energía empleada para su transmisión. El procesado de la señal captada por los sensores resulta imprescindible para enviar sólo aquellos datos estrictamente necesarios a través de la red.

La estructura general del mensaje de muestreo debe incluir los siguientes campos para identificar correctamente una medición:

- ID del nodo
- ID del sensor
- Hora de la medición
- Valor de la medición

De la misma forma, si se envía a un nodo sensor un comando de actuación para que éste aplique alguna acción, el mensaje debe contener los siguientes elementos clave:

- ID del nodo
- ID del actuador
- Valor de la acción a aplicar

3.4. Comunicaciones

La topología de la red dependerá de varios parámetros: área de influencia, número de nodos, disponibilidad de energía o interacción con otras redes de comunicaciones, como Internet, son algunas. En cualquier caso, el modo de funcionamiento API de los módulos XBee asegurará las comunicaciones de forma fiable sin importar la topología

Para el buen funcionamiento de las comunicaciones deberán considerarse aspectos como interferencias y ruido que puedan afectar el buen funcionamiento. Optimizar el funcionamiento de las comunicaciones, reduciendo el número de saltos entre nodos que la información debe realizar hasta llegar a su destino y limitando las comunicaciones a unos determinados momentos de día ayudará a reducir el tráfico y el consumo energético de la red.

Garantizar la seguridad de la red es necesario para evitar intromisiones e interferencias que puedan afectar su funcionamiento. Por una parte la modulación DSSS que utilizan los módulos por herencia de la norma IEE 802.15.4, tema del que se habló en el apartado anterior, resulta muy difícil de interceptar, lo que hace de estas radios un medio de comunicaciones muy seguro. Por otro lado, los módulos Xbee utilizan el sistema de encriptación AES de 128 bits, un esquema de cifrado muy fiable que ha sido aprobado para su uso en el cifrado de documentos oficiales por parte de la NSA (National Security Agency). Por estos motivos no se consideraron medidas adicionales sobre esta materia en la realización de este proyecto.

3.5. Sincronización temporal

Teniendo en cuenta la velocidad de transmisión de datos del protocolo ZigBee (250 kbps), y que la extensión de los mensajes no será mayor a unos cuantos bytes, el tiempo de recorrido de los mensajes será muy corto. Aun en redes grandes el posible desfase entre el momento del muestreo en el nodo sensor y el arribo al nodo central no superará será inferior a unos pocos segundos., siempre que no exista un excesivo tráfico en la red.

Por ello se podría asumir en la mayoría de los casos el momento de recepción de los datos de muestreo en el nodo central como el momento de muestreo, sin que esto suponga un error significativo. Aún así se podría incorporar a los nodos sensores un módulo que registre de forma exacta el momento en que se realizaron las medidas para mayor precisión.

Otra razón para incorporar este tipo de módulos de reloj en los nodos es sincronizar las comunicaciones de la forma más eficiente para optimizar el consumo de energía. Si todos los nodos conocen la hora actual, se pueden programar las comunicaciones para que se realicen sólo en momentos concretos, manteniendo apagados los dispositivos de comunicación (que son los que más energía consumen) el resto del tiempo y ahorrando así energía.

3.6. Energía

Un punto muy importante en las WSN es el consumo de energía. Se espera que los nodos consuman la menor cantidad de energía posible tanto para la recogida, el procesamiento de datos y las comunicaciones. La administración de la energía se vuelve, entonces, un punto muy importante a estudiar sobre todo cuando los nodos tienen que funcionar en entornos aislados o donde no un suministro continuo disponible.

La selección correcta de los componentes es fundamental para el buen uso de la energía. Si bien se explicó que el protocolo ZigBee incorpora muchas características para reducir el consumo de las radios, optimizar el funcionamiento de la red será fundamental. Por otro lado, la selección del micro controlador y del resto de componentes electrónicos y la optimización de su consumo también son muy importantes. Configurar los componentes electrónicos en sus modos de ahorro de energía será muy útil a la hora de disminuir el consumo.

Conectar a los nodos a la red eléctrica mediante algún adaptador que permita mantener las radios encendidas en todo momento simplificará la implementación. Esto permitirá ahorrar espacio de memoria en los dispositivos electrónicos y disponer de una red de comunicaciones más dinámica, una característica fundamental muy deseable en redes destinadas a la automatización y el control en general.

El uso de baterías de polímero de litio o "Lipo" aseguran mantener un tamaño reducido, gran densidad de carga, baja descarga y bajo coste. Incorporar una fuente de generación de energía, como paneles solares o aerogeneradores es una buena opción para recargar la batería e incrementar la autonomía del dispositivo.

3.7. Software

A la hora de desarrollar el software hay que pensar en tres apartados:

- Software del nodo sensor: los micro-controladores Arduino utilizan su propio lenguaje para escribir las aplicaciones que ejecutan. Éste está basado en el lenguaje Wiring, el cual desciende, a su vez, de Processing. Todos estos son lenguajes de programación de código abierto muy fáciles de conseguir, aprender e implementar. La posibilidad de modificar el programa de los nodos a través de la red, de forma inalámbrica resulta más cómoda y de mucha utilidad si el acceso al nodo resulta complicado.

- Software del módulo XBee: el fabricante de los módulos de comunicación ofrece la aplicación X-CTU para la configuración de los dispositivos. Este programa permite configurar los diferentes parámetros de los módulos: su rol en la red, las conexiones con otros dispositivos, parámetros energéticos, y descargar actualizaciones del firmware para tener siempre la versión más reciente instalada.
- Software de procesado de datos: Este programa se encarga de la operación de la red en cualquiera que sea la aplicación a la que se destine. Debe poder almacenar y modificar características de la red de sensores como número y tipo de nodos, sensores, actuadores, muestreo, comunicaciones y demás. También se encargará de administrar los datos recibidos. Como se especificó en la definición del proyecto, se propone utilizar el lenguaje Processing para esta tarea.

Una vez conocidas las características generales y las prestaciones requeridas para alcanzar los objetivos del proyecto, en los siguientes capítulos se abordarán los apartados de diseño: selección del hardware, confección de los nodos, diagramas de operación, programación del software y pruebas de funcionamiento.

4.Hardware

4.1. Material seleccionado

En primer lugar se analizará el material seleccionado para desarrollar el proyecto. Como se dijo en el apartado anterior, el proyecto aquí desarrollado intenta servir de base para la implementación de las WSN, dado que la elección final deberá ajustarse a las necesidades del usuario para disminuir los costes y maximizar las prestaciones. Los componentes seleccionados no sólo responden a las prestaciones requeridas sino también a las necesidades propias del proyecto para realizar ensayos.

4.1.1. microcontrolador

Para recoger datos y realizar acciones los nodos necesitan disponer de un microcontrolador. La misión de este componente será, por un lado, captar las señales de los sensores y procesarlas para obtener una medición; y por otro, emitir señales de control para los actuadores.

Como se especificó en la definición del proyecto, el microcontrolador seleccionado para esta aplicación es un modelo proveniente de la familia de microcontroladores Arduino. Esta elección responde, como ya se dijo a su facilidad de uso, entorno libre de desarrollo, la existencia de una comunidad de usuarios muy activa a través de Internet y su reducido coste.

En concreto se ha seleccionado el modelo UNO. Se trata del modelo más popular de la familia Arduino ya que contiene todos los elementos característicos de Arduino: layout de los pines, conexión USB B con conversor USB-serie y jack de alimentación con regulador de voltaje.

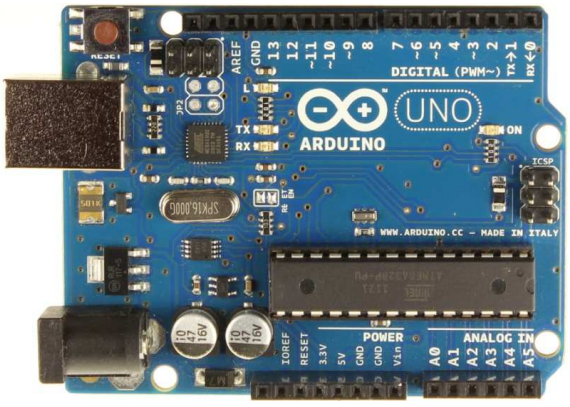

Figura 7. Arduino UNO. Fuente: Arduino.

El corazón del UNO es el ATmega328p, un microcontrolador de 8 bits con velocidad de reloj de 16 MHz desarrollado por ATMEL. Es un componente muy robusto con capacidad de 32 kB de memoria flash programable y 2 kB de SRAM. Estos datos son muy importantes a la hora de desarrollar el proyecto ya que son factores limitantes. No sólo debe caber todo el software de control de los nodos sensores, sino que debe de quedar suficiente espacio para la aplicación que se implemente funcione correctamente.

Tabla 2. Características principales del Arduino UNO rev.3.

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (rec)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

En el anexo de hojas de datos correspondiente se pueden leer más detalles sobre el microcontrolador ATmega328p.

4.1.2. XBee

Las comunicaciones de la red diseñada en este proyecto se llevarán a cabo mediante módulos XBee bajo protocolo ZigBee, como se especificó anteriormente. Esto se debe a que es la elección predilecta de los usuarios de Arduino para trabajar con radios bajo protocolo IEEE 802.15.4 en general y ZigBee en particular. Por ello, el volumen de información disponible acerca de su implementación es enorme.

Para este proyecto se seleccionaron los módulos de mayor potencia disponible de la Serie 2, los XBee-Pro S2B, con antena RP-SMA. Este módulo transmite en la

banda de 2,4GHz a una velocidad de 250 Kbps. Dispone de un transmisor de 10mW (versión internacional), que le otorga un alcance de hasta 1500m al aire libre.

Figura 8. Módulo XBee-PRO S2B con antena RP-SMA. Fuente: MCI electronics Ltda.

Este componente, aunque más costoso que su equivalente de baja potencia, permite ajustar la potencia de emisión del transmisor, con lo que se pueden realizar pruebas para estudiar el consumo energético y el alcance la señal transmitida. Además, la antena RP-SMA, aunque más voluminosa que el resto de opciones: chip, whire whip o U.FL connector, es un tipo de conector muy robusto y extendido que permite conectar el transmisor fácilmente a antenas y otros dispositivos como amplificadores para mejorar la recepción.

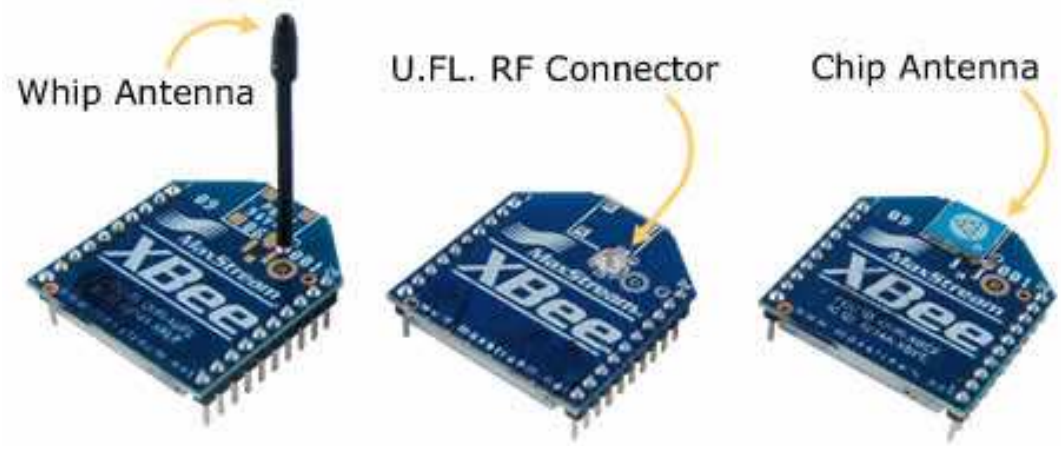

Figura 9. Diferentes antenas disponibles para módulos XBee. Fuente: MCI electronics Ltda.

Cabe destacar que no existen mayores diferencias entre los módulos XBeee de la Serie 2, por lo que todos ellos son básicamente intercambiables.

Tabla 3. Características principales del módulo XBee-PRO S2B seleccionado.

Modelo	XBP24BZ7SIT-004J
Velocidad de transmisión	250 Kbps
Frecuencia de transmisión	2,4 GHz
Alcance en interiores/urbano	40 m
Alcance al aire libre sin interferencias	1500 m
Potencia de transmisión	10mW (+10 dbm)
Sensibilidad del receptor	-102 dbm
ADC inputs	4 (10 bits)
Digital I/O	10
Encriptación	128 bits AES
Metodos de configuración	Comandos API o AT locales u "over-the-air"
Voltaje de suministro	2.7-3.6 VDC
Corriente de transmisión	205 mA

Los módulos XBee se comunican con Arduino a través del puerto serie nativo de esta última. Su implementación es tan sencilla que prácticamente pueden ser transparentes para el microcontrolador. A pesar de ello, hacer uso de las características avanzadas del módulo XBee requerirá de una interacción más compleja, el modo de funcionamiento API. Sobre este tema se hablará en el apartado de software.

A la hora de implementar el módulo XBee con Arduino hay que tener en cuenta que los dispositivos operan con voltajes diferentes. El Arduino UNO utiliza el procesador descrito anteriormente que opera con lógica de 5V, mientras que los módulos XBee operan con lógica de 3,3V. Conectar directamente una línea de datos del Arduino al XBee podría dañar este último, y pasar datos directamente desde el XBee al Aduino podría no ser posible al no poder leer el procesador del UNO un voltaje de 3,3V como un nivel alto. Para solucionar este problema y permitir la comunicación entre los dispositivos habrá que considerar la incorporación de un convertidor de niveles lógicos.

En el anexo de hojas de datos correspondiente se pueden leer más detalles sobre las especificaciones del módulo XBee seleccionado.

4.1.3. Energía

En este proyecto se utilizarán tres formas de alimentación para los nodos de la red. La primera es utilizar la energía proveniente del puerto USB al que se encuentra conectado el nodo. Ésta será la alimentación más común para el nodo coordinador, que estará conectado casi siempre al sistema de procesado final de datos. Aunque esta es una forma segura y constante de suministrar energía, se deben de tener en cuenta las limitaciones eléctricas de los puertos USB, que suministran por defecto un máximo de 100 mA a 5V.

Figura 10. Cable USB A-B utilizado en el proyecto. Fuente: propia.

La segunda forma de alimentación considerada es la que utilizarán los nodos de tipo routers y, en general, cualquier nodo de la red que deba y pueda mantenerse siempre encendido. Se trata de un adaptador AC/DC que suministra hasta 1A a 9V. Es ideal para utilizar con Arduino ya que posee un jack macho de idénticas características al que disponen las placas UNO.

Figura 11. Adaptador AC/DC de 9V utilizado en el proyecto. Fuente: ebay.

La tercera opción estudiada en este proyecto para alimenta los nodos es un paquete de pilas recargables conectadas mediante un portapilas al jack de la placa Arduino. Ésta será la forma de alimentación principal para aquellos nodos

que deban funcionar de forma autónoma durante largos periodos. En este proyecto se utilizarán pilas recargables AA de NiMH de 2300 mAh a 1.2V por unidad, en un portapilas de 6 unidades. De esta manera, el paquete completo suministrará unos 2300 mAh a 7,2 V, valor muy próximo al voltaje mínimo recomendado para alimentar las placas Arduino.

Figura 12. Pila recargable y portapilas utilizado en el proyecto. Fuente: RS Components y ebay.

4.1.4. Wireless SD Shield

El módulo XBee requiere de un adaptador para su utilización con las placas Arduino no sólo para adaptar los niveles lógicos, como se explicó antes, sino también sus pines. Todas las radios XBee utilizan pines con 2mm de separación, pero las placas Arduino, las protoboards y las placas de prototipado utilizan pines con 0,1" (2,54 mm) de separación. Esto hace imposible conectarlos directamente. Además, un adaptador es necesario para poder conectar y configurar las radios a través de su aplicación X-CTU.

En Internet se pueden encontrar multitud de adaptadores y placas de conexión para solucionar el problema, entre los que destaca el XBee Explorer desarrollado por Sparkfun. Pero en este proyecto se optó por una solución más integral, pensada para trabajar con Arduino: la Wireless SD Shield desarrollada por el Team Arduino. Se trata de una de las tantas placas especialmente diseñadas para encajar en el layout de Arduino, que dispone un socket de conexión para módulos XBee. Esta placa incluye todos los demás componentes para adaptar los niveles eléctricos y trabajar cómodamente, con lo que no son necesarios componentes adicionales.

Además de esto la placa incluye un socket micro SD y también todos los componentes electrónicos que nos permitirán trabajar fácilmente con tarjetas de

almacenamiento. Finalmente, la placa dispone de un pequeño espacio de prototipado para soldar y conectar componentes.

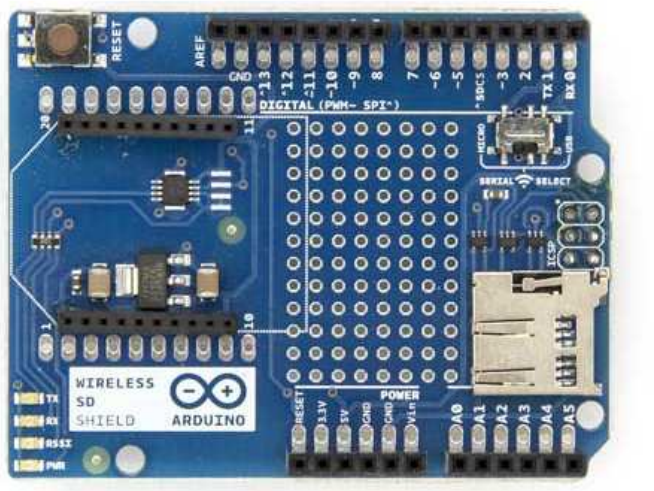

Figura 13. Wireless SD Shield. Fuente: Arduino.

Esta palca dispone de un muy útil interruptor que permite cambiar el destino de las comunicaciones. En una posición la radio se comunica con el procesador ATmega328p, y en la otra, con el conversor USB-serie de la placa Arduino. Como resultado esta shield permite utilizar el Arduino como un adaptador para conectar las radios a un ordenador y configurarlas.

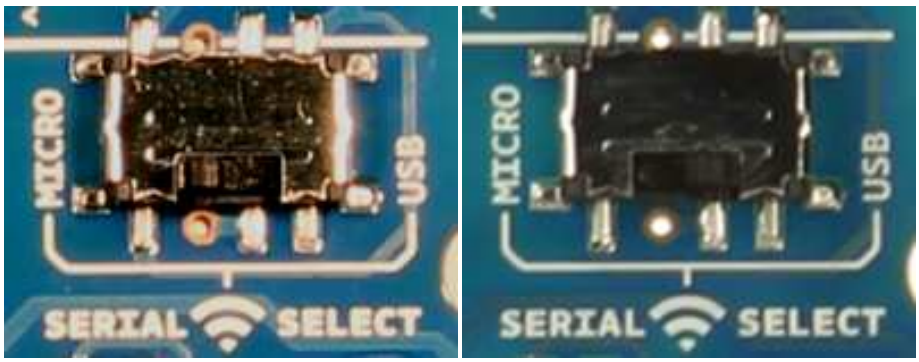

Figura 14. Aspecto del conmutador de la shield en las dos posiciones.

4.1.5. Otros

4.1.5.1. Almacenamiento

Aprovechando el socket micro SD de la Shield seleccionada, se ha considerado conveniente implementarlo. Una posible aplicación sería almacenar los datos de los nodos sensores en el nodo central, funcionando la red de forma aislada y este nodo como compuerta para volcar los datos en otra red mayor, como Internet

Se ha seleccionado una memoria de tipo micro SD estándar para esta tarea. Estas memorias por especificación técnica deben ser compatibles con el estándar de comunicaciones SPI (Serial Peripheral Interface, un estándar utilizado para comunicación entre circuitos integrados en equipos electrónicos). Este protocolo es ampliamente utilizado en muchos microcontroladors y en particular en Arduino.

En particular se han seleccionado memorias micro SD por con una capacidad de almacenamiento máxima de 2GB. La razón de esto último radica en que estas tarjetas pueden formatearse con el sistema de archivos FAT 16, lo que permite implementarlas en Arduino con librerías simplificadas y más ligeras (que ocupan menos espacio de memoria). Teniendo en cuenta que los datos almacenados consistirán básicamente en archivos de texto plano, el tamaño de la memoria es más que suficiente y esto no supondrá una desventaja apreciable.

Figura 15. Tarjeta micro SD junto a un adaptador.

4.1.5.2. Sincronización horaria

Cuando el nodo sensor funcione en modo off-line necesitará de un reloj lo suficientemente preciso para registrar las mediciones correctamente en el tiempo y para sincronizar las comunicaciones y la transferencia de datos haciendo un uso eficiente de la energía, como se explicó en el apartado de prestaciones.

En los modos de ahorro de energía más eficientes el Arduino apaga sus circuitos osciladores, lo que le impide funcionar como reloj. Además, la programación ocuparía un ya de por si limitado espacio de memoria. Por su parte, el mósulo XBee tiene la capacidad de mantener un temporizador funcionado cuando entra en modo SLEEP, pero no tiene la capacidad de almacenar la hora y fechas actuales.

Por esta razón se ha considerado oportuno incorporar un último módulo electrónico al nodo sensor. Se trata de un reloj a tempo real o RTC (Real Time

Clock, por sus siglas en inglés). Estos componentes son pequeños circuitos electrónicos capaces de mantener y almacenar la hora y fecha actuales, y de transmitirla cada vez que se le solicita. Para ello disponen de sus propios circuitos osciladores y algunos incorporan baterías con el fin de mantener el reloj funcionando aunque se corte el suministro primario de energía.

El módulo RTC elegido en este proyecto es el Tiny RTC, que incorpora el reloj a tiempo real en su circuito integrado DS1307 de Maxim Integrated.

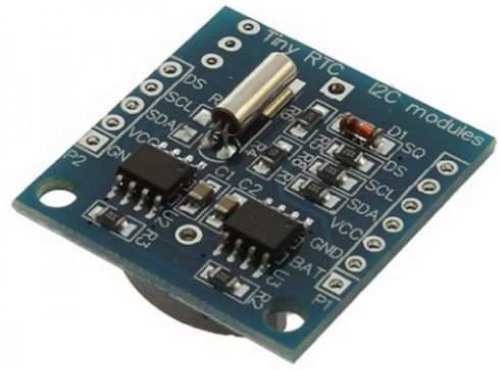

Figura 16. Módulo Tiny RTC con el circuito DS1307. Fuente: eBay.

Este módulo utiliza comunicación I2C (Inter-Integrated Circuit), que es otro protocolo de comunicaciones entre circuitos integrados, y puede ser implementado fácilmente en Arduino gracias a las librerías especialmente creadas por la comunidad de usuarios durante los últimos años.

4.2. Confección de los nodos

A continuación se explicará cómo se armaron los nodos de la red: los tipos de nodos existentes: sensor, router y coordinador; sus funciones, y los componentes necesarios que deben incorporar para llevarlas a cabo.

4.2.1. Nodo sensor

El nodo sensor es el encargado de recoger los datos de los sensores conectados a éste y de comunicarlos, a través de la red inalámbrica, al nodo central donde serán procesados. Estos nodos también pueden tener la capacidad para realizar alguna acción, ya sea de forma independiente o por orden del nodo central.

Teniendo en cuenta los modos de funcionamiento explicados en el apartado de prestaciones y los componentes seleccionados, se plantea una configuración general del hardware para la confección de los nodos sensores.

La recogida y el procesado de los datos es tarea del Arduino, mientras que el módulo XBee se encargará de las comunicaciones del nodo. La comunicación entre los dos módulos se realizará utilizando la Wireless SD Shield como adaptador.

Dado el poco tiempo de tránsito de los mensajes por la res, explicado en el capítulo anterior, no se ha considerado necesario finalmente incorporar el módulo RTC en esta etapa del proyecto. Tampoco se ha considerado necesario incorporar el uso del módulo SD de los nodos. Teniendo en cuenta que la idea es que los nodos estén conectados a la red siempre sea necesario, los datos se pueden recoger y transmitir directamente sin necesidad de almacenarlos. Una razón adicional para descartar el uso de estos dos componentes fue el enorme espacio de memoria que ocupaban. De acuerdo a las primeras pruebas, se utilizaba alrededor de un 50% de la memoria flash y un 64% de la RAM de Arduino sólo para las librerías y el código de estos dos componentes en un programa sencillo. De todas formas, la utilidad de estos componentes podría justificar su implementación en futuras evoluciones del proyecto.

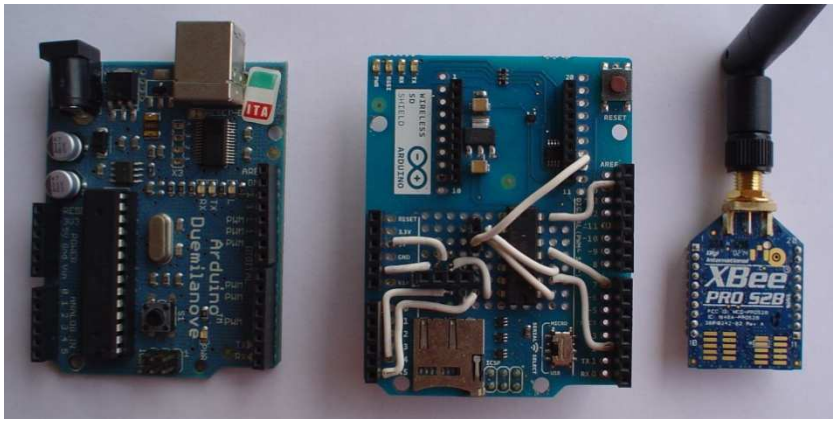

Figura 17. Componentes utilizados para confección del nodo sensor.

Dependiendo de la frecuencia de muestreo, será mejor alimentar el nodo con un adaptador como el seleccionado u otro elemento que asegure un suministro continuo. El uso de pilas o baterías sería recomendable en entornos aislados o lejos de un enchufe, y siempre que se disponga de alguna fuente de energía para recargar las baterías, como paneles solares. Esto no sería un problema tan importante en el caso de activación por excepción ya que en estos casos el nodo sensor sólo se enciende y transmite cuando se activa el sensor por un estímulo externo, como en el caso de una alarma. En estos casos sería posible alimentar el nodo con pilas cómodamente.

Con el fin de aprovechar los modos de ahorro de energía tanto de Arduino como de XBee, fue necesario incorporar un último componente. Se trata de un circuito inversor lógico (sn74ls04 de Texas Instruments) que transforma la señal de encendido del módulo XBee, en el pin 9 del módulo, en una señal para ejecutar en una señal de interrupción, en el pin 2 de la placa. De esta forma cuando el módulo se enciende la señal saca al Arduino del modo SLEEP al recibir una interupción de nivel bajo; mientras que al apagarse, la señal recibida de nivel lógico alto devuelve al Arduino el modo de ahorro de energía.

Dada la mayor facilidad de programar y de reconfigurar los parámetros de funcionamiento a través de la red, se optó por otorgar al módulo XBee el control del nodo. Las razones de esto serán ampliadas en el siguiente apartado.

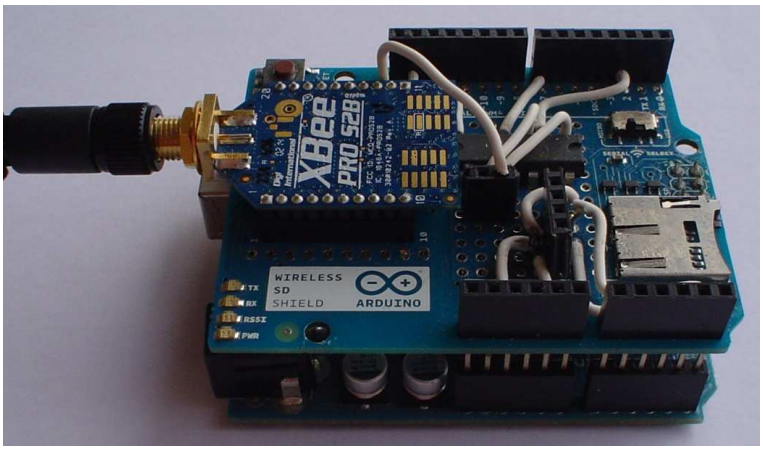

Figura 18. Aspecto del nodo sensor montado.

4.2.2. Nodo router

De acuerdo a lo explicado sobre el protocolo ZigBee, todo nodo de tipo *End device* requiere de un dispositivo que actúe como su padre y lo ayude a conectarse a la red. Éste puede ser el coordinador de la red o bien una radio configurada como router. Los nodos configurados como routers tienen la capacidad de pasar mensajes provenientes de otros nodos hacia su destinatario final.

Este tipo de nodos permite incrementar el alcance y el área de cobertura de la red, aunque en contrapartida no tienen la capacidad de entrar en modo de ahorro de energía ya que deben mantenerse alerta en todo momento. La presencia de nodos de este tipo en una WSN dependerá de las características de la propia red.

La construcción básica de estos nodos requeriría simplemente de un módulo XBee, ya que su función básica es servir de puente a los nodos sensores para comunicarse con el coordinador. Este nodo deberá de disponer de una fuente de alimentación que permita mantener el nodo encendido en todo momento.

También cabe la posibilidad de utilizar un nodo router para realizar mediciones que, por las características de las comunicaciones, debería tratarse como un nodo sensor. En este caso la construcción sería igual a la de aquellos y sólo se modificaría la configuración del módulo de comunicaciones.

4.2.3. Nodo coordinador

Toda red ZigBee requiere de un nodo coordinador para crear y administrar la red. Este nodo es el principal destino de toda la información recogida y el origen de cualquier instrucción o acción correctora. Es el nodo a través del cual el operador de la red tendrá acceso a ésta. Teniendo en cuenta lo anterior, el nodo central de la red será un sistema informático conectado al XBee configurado como coordinador.

Según esto el nodo coordinador necesitará se construirá con su correspondiente módulo XBee y el adaptador para conectarlo al ordenador de procesamiento de datos. La alimentación se realizará en este caso a través del puerto USB al que se encuentra conectado el XBee. En este caso el nodo estará conformado por el módulo XBee utilizando como adaptador la Wireless SD Shield, con el conmutador en la posición USB, conectada a la placa Arduino sin el MCU.

Figura 19. Aspecto de la placa Arduino sin el MCU.

Habiendo explicado la constitución física de los nodos de la red, en el siguiente apartado se abordará la programación de la red: la configuración de los diferentes módulos para su funcionamiento, los diagramas de operación de los nodos y el software de volcado y procesamiento de datos.

5. Software

Objetivo: Ocupar el menor espacio posible con el programa del algoritmo de funcionamiento del nodo para dejar el mayor espacio posible para la aplicación

5.1. API Frames

A lo largo de los apartados anteriores se ha comentado que las comunicaciones se realizarían con los módulos XBee en modo API, y llegados a este punto y antes de comenzar a describir la solución planteada para el software resulta necesario explicar qué son, para qué sirven y cuáles son sus ventajas. API son las iniciales de Application Programming Interface (Aplicación de programación de la interface) y se trata de una serie de interfaces estándares creados para facilitar la comunicación entre un programa de software con otro.

Las radios XBee se comunican entre ellas mediante el protocolo ZigBee a través de radiofrecuencias, pero cuando las radio tienen que comunicarse con un sistema informático que funcione como *Host*, como la placa Arduino o el ordenador central de recogida de datos, utilizan su UART o puerto serie nativo. Existen dos modos de funcionamiento para realizar esta comunicación llamada de radio local: transparente/ de comandos o API.

El primero es el más sencillo de implementar. Básicamente se comporta como el típico enlace de comunicación serie entre dos sistemas informáticos, mientras que todo el trabajo de comunicación entre las radios resulta ajeno, transparente al host, quien ignora cómo se realiza. Otra ventaja añadida es la fácil comprensión de los mensajes, que son transformados a caracteres ASCII imprimibles, para las personas. En Arduino esto se realiza mediante los comandos `Serial.print()` y `Serial.println()`. A pesar de esto, este modo de funcionamiento presenta unas cuantas desventajas. El principal es que todo mensaje que se envíe de esta manera a través del UART cae metafóricamente a un agujero negro. El host no puede más que enviar su mensaje y no tiene garantía ninguna de que éste alcance el destinatario o de que haya sido recibido por la radio local para transmitir. Otra desventaja del modo transparente es que el host no puede indicarle a la radio un destinatario concreto, todos los mensajes se enviarán a aquella radio enlazada con la local. Esto hace imposible crear redes malladas, donde el mensaje que se genera en un extremo de la red debe alcanzar el otro, teniendo que realizar múltiples saltos entre radios routers por el camino. Además, como todos los datos recibidos se interpretan

como una seguidilla de caracteres ASCII, el tratamiento de los datos numéricos en aplicaciones requiere de un paso previo de reconstrucción de los números a partir de los dígitos con funciones como `atoi()` (ASCII to integer)

Los API son una forma de comunicación específicamente diseñados para comunicar dos sistemas informáticos. Cuando una radio XBee está configurada en modo API, toda comunicación con ésta se realiza mediante uno de los llamados API frame. Un API frame o Marco API es una estructura de comunicaciones formada por una serie de bytes que delimitan el mensaje, su tamaño, tipo, origen y la propia información a transmitir. Todo marco API posee una estructura básica que comienza con un byte delimitador del mensaje, éste byte en notación hexadecimal es el 0x7E (cuyo equivalente en ASCII es "~" o el carácter 126 de la tabla. La notación hexadecimal es típica a la hora de tratar con bytes ya que facilita la interpretación de los números binarios que representan, y es la que se utilizará para describirlos en la mayoría de apartados de este proyecto. En todo caso cabe recordar que un hexadecimal representará aquí un número de 8 bits). A éste le siguen dos bytes que delimitan el tamaño del mensaje, indicando el número de bytes contenidos en el Frame data. El Frame data es la parte en la que se incluye el tipo de mensaje que se recibe o transmite y la propia información en a transmitir. Finalmente, el último Byte es el llamado Checksum. Se trata de un byte de comprobación que se calcula a partir de los datos del mensaje y que asegura la integridad de los datos: si el número recibido es diferente del esperado significa que los datos no han sido transmitidos correctamente. En el manual del XBee ZB desarrollado por DIGI se explica cómo calcularlo.

Tabla 4. Estructura general de un marco API. Fuente: [5]

Start delimiter	**Length**		**Frame data**		**Checksum**
Byte 1	Byte 2	Byte 3	Byte 4…	…Byte n	Byte n+1
0x7E	MSB	LSB	Api specific structure		Single byte

Existe más de una docena de marcos y cada uno responde a una tarea en concreto, como envío de datos, recepción de datos, configuración de parámetros de la radio, estado del mensaje enviado o estado de la radio. Todo frame data comienza por un byte que indica el Frame type o tipo de mensaje del que se trata, como 0x10 para solicitar una transmisión o 0x90 para indicar un mensaje entrante.

Tabla 5. Lista con los Frame Type más utilizados en este proyecto.

Frame type	Función	Descripción
0x10	TX request	Solicitud de envío de mensaje de datos.
0x90	RX received	Mensaje de datos recibido o entrante.
0x8B	TX response	Estado del mensaje enviado.
0x08	AT command	Solicitud de configuración AT de la radio local.
0x88	AT command response	Respuesta recibida a un mensaje del tipo anterior.
0x17	Remote Command Request	Solicitud de configuración AT de un radio remota.
0x97	Remote Command Response	Respuesta recibida a un mensaje del tipo anterior.
0x8A	Modem Status	Mensaje recibido que indica el estado de la radio.

5.1.1. Ejemplo

Se desea transmitir el mensaje "HOLA MUNDO" desde un nodo end device al nodo coordinador de la red.

Toda radio XBee dispone de un número de identificador único en el mundo, se trata de su dirección de 64 bits. Ésta se suele dividir en dos partes de 32 bits, la alta y la baja. Actualmente todas las radios XBee comparten la misma parte alta (0x0013A200), mientras que la parte baja es diferente. Teniendo en cuenta el tamaño de la dirección, habría disponibles más de $1{,}88 \cdot 10^{19}$ radios sin que se duplique una dirección, lo cual es un número exageradamente grande. Pero incluso teniendo en cuenta la parte baja se pueden construir más de $4 \cdot 10^{9}$ radios con dirección única. Además de esta dirección, cuando una radio se enlaza a una red, el coordinador le asigna automáticamente una dirección de 16 bits única para cada dispositivo de la red. Esta dirección facilita el direccionamiento de los mensajes y acelera las comunicaciones.

En este caso supondremos que la dirección del emisor es: 0x0013A200 0x40BABC69, mientras que la del coordinador es 0x0013A200 0x40BABC78. De acuerdo a las fuentes consultadas [5] el marco de transmisión del mensaje sería el que sigue:

Tabla 6. Marco API para la transm isión del mensaje "HOLA MUNDO".

Campos del marco		Offset	Ejemplo	Descripción
Start Delimiter		0	0x7E	Delimitador de mensaje
Length		1	0x00	Longitud del Frame data: 24 bytes.
		2	0x18	
Frame data	Frame type	3	0x10	Tipo de marco: transmit request
	Frame ID	4	0x01	ID propio del mensaje
	64 bit destination address	5	0x00	Dirección de 64 bits del coordinador de la red. También se podría haber utilizado la dirección comodín que identifica al cooridnador: 0x00000000 0x00000000
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0xBA	
		11	0xBC	
		12	0x78	
	16 bit destination address	13	0x00	Dirección de 16 bits por defecto del coordinador
		14	0x00	
	Radius	15	0x00	Parámetros de configuración de la transmisión
	Options	16	0x00	
	RF data	17	0x48	"HOLA MUNDO" escrito en sus bytes componentes para cada carácter, de acuerdo la tabla ASCII
		18	0x4F	
		19	0x4C	
		20	0x4C	
		21	0x20	
		22	0x4D	
		23	0x55	
		24	0x4E	
		25	0x44	
		26	0x4F	
Checksum		27	0x45	

La principal desventaja de los API frames a la hora de implementar este modo de comunicación, como puede verse en la tabla anterior, es su complejidad que obliga a programar funciones y/o librerías para construir y descifrar los marcos utilizados en la comunicación. Además, todos los datos deben ser enviados como bytes, así que deben ser convertidos independientemente de su formato de origen.

Pero todo esto se compensa por las enormes ventajas. En modo API las radios pueden formar una red mallada, con la seguridad de que los datos enviados alcanzarán su destino, gracias al direccionamiento de los mensajes, los mensajes de estado de la transmisión y de estado de la radio que nos permitirán transmitir sólo cuando la radio indique que se encuentra conectada a la red. Para más información sobre los marcos API pueden leerse las referencias consultadas. [5] [7]

5.2. Software de configuración de las radios: X-CTU

El productor de las radios XBee, DIGI, ofrece a través de su sitio web la aplicación X-CTU para configurar los dispositivos. Este programa en su versión más reciente (6.1.1) resulta muy sencillo de utilizar, gracias a una interfaz mejorada, y resulta casi irreemplazable por las utilidades que ofrece, como una herramienta para generar marcos API y un terminal para comunicarse directamente con las radios, o una herramienta de búsqueda de radios automatizada que permite encontrar e incorporar automáticamente a la red nuevas radios en modo API.

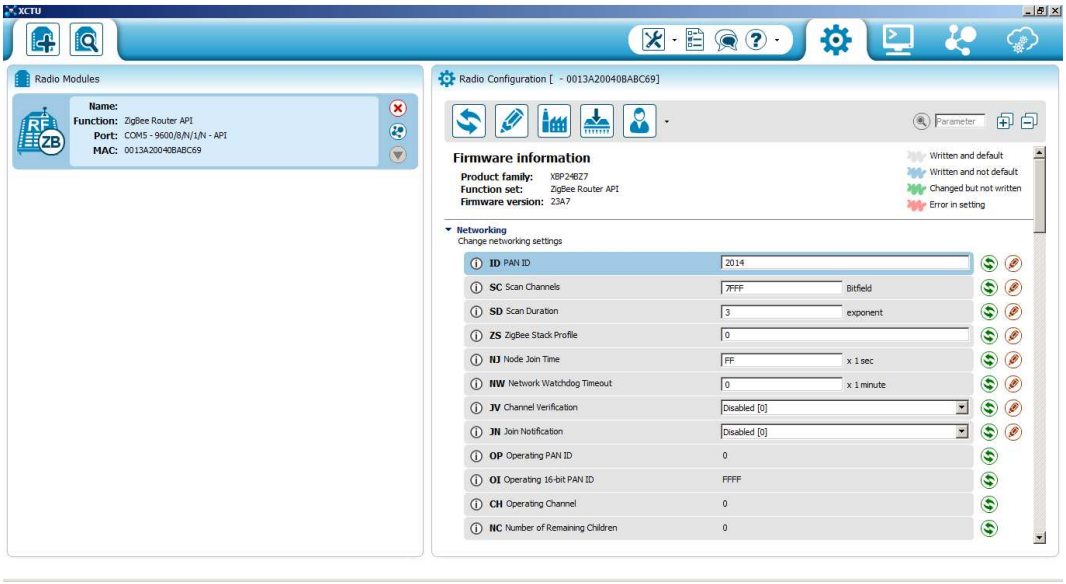

Figura 20. Pantalla de X-CTU en modo de configuración. Fuente: propia.

La configuración de las radios XBee para ser empleadas en este proyecto se realiza mediante utilizando como adaptador la placa Arduino y la Wirelesss SD Shield, como se explicó en el capítulo anterior. Para poder configurar las radios cómodamente se colocará el conmutador de la Shield en la posición USB, de esta manera el UART del XBee estará directamente conectado con el conversor USB-TTL de la placa Arduino. Por otro lado resulta más que recomendable extraer el microcontrolador de la placa Arduino, para evitar posibles interferencias entre éste y el XBee. Aunque inicialmente se realizaron pruebas cargando un programa vacío en el Arduino, a la hora de trabajar en modo API, incluso en estas condiciones el MCU interfería en el proceso.

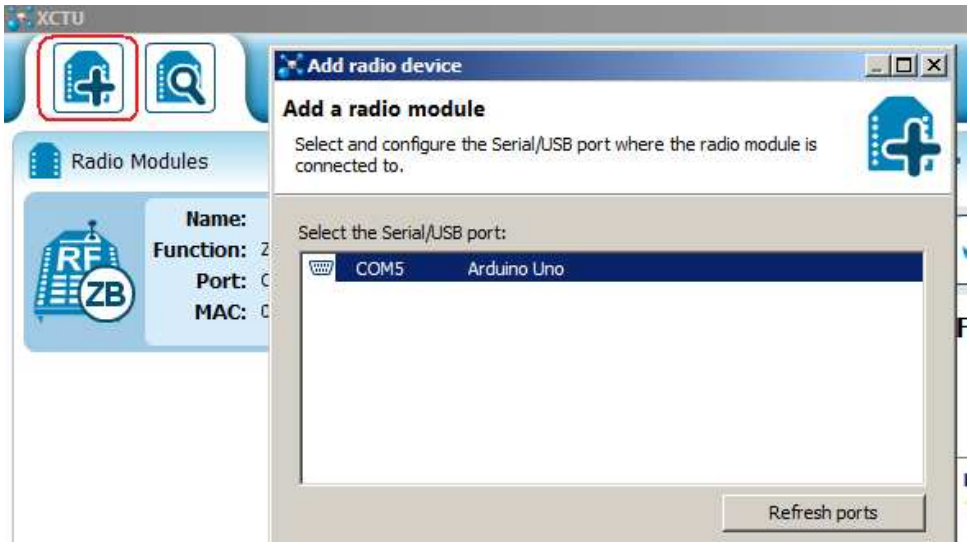

Figura 21. Ventana de agregar dispositivo. Fuente: propia.

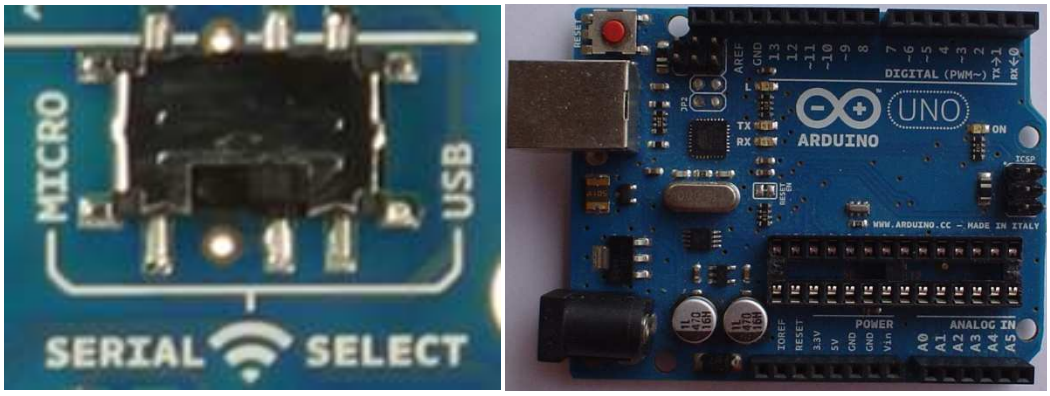

Figura 22. Interruptor de la Shield en la posición USB y placa Arduino sin el MCU.

Una vez conectado al ordenador y añadido al programa X-CTU, habrá que cargar el firmware para la radio. Esto sólo se puede hacer desde esta aplicación. Para ello se seleccionará el tipo de radio, el rol que la radio tendrá en la red (coordinador, router o end device) y siempre en modo API y la versión del firmware. Algunas veces durante la carga del firmware el programa solicita resetear la radio, paso que se ejecuta puenteando el pin 5 del XBee a GND.

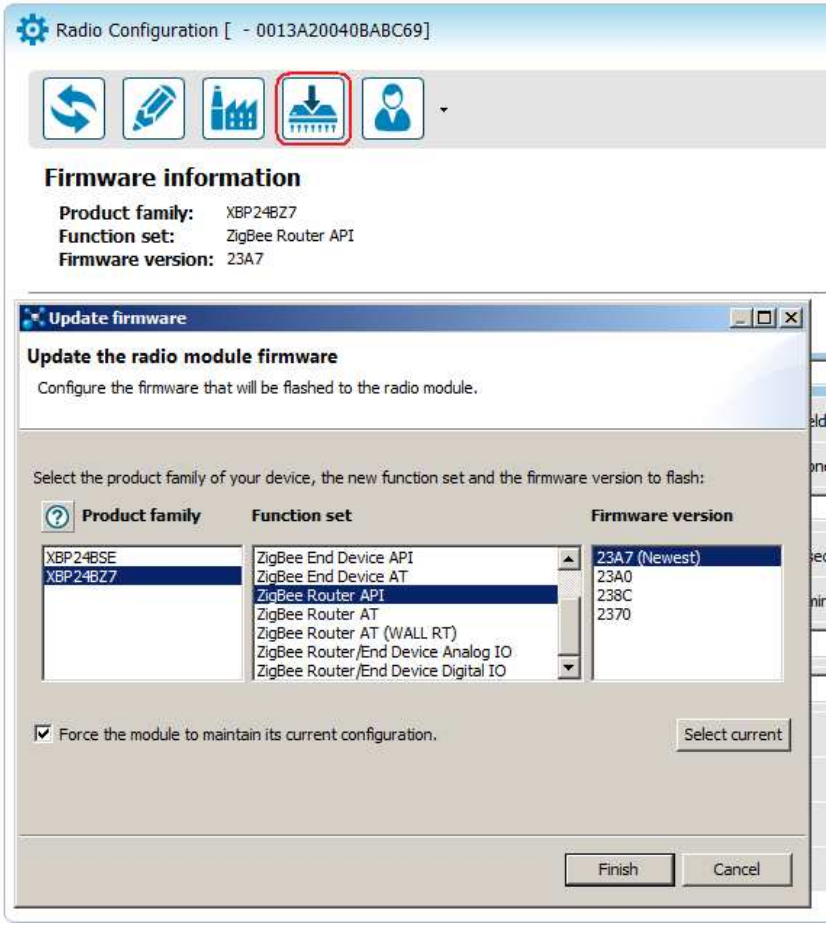

Figura 23. Ventana de carga del firmware en X-CTU. Fuente: propia.

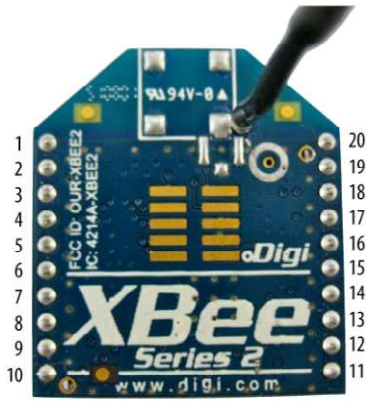

Figura 24. Disposición de pines del XBee. Fuente: [5]

En la pestaña de configuración, los parámetros a tener en cuenta son los de direccionamiento: ID de la red, dirección del dispositivo principal al que conectarse y los parámetros del modo SLEEP para habilitar el modo de ahorro de energía. Un parámetro importante a tener en cuenta es el AP que deberá configurarse con un valor de 01. Este parámetro se configurar, también con el valor 2, conocido como escaped. En esa configuración se incluyen características adicionales de fiabilidad en las comunicaciones. A pesar de ello supone una complejidad adicional trabajar en este modo que no resulta justificable en este proyecto, ya que hay una serie de bytes reservados para comunicaciones que si se reciben en una posición diferente, son transformados mediante operaciones bitwise que al ser recibidos deben de revertirse. Por ello se trabajará siempre en el modo simple.

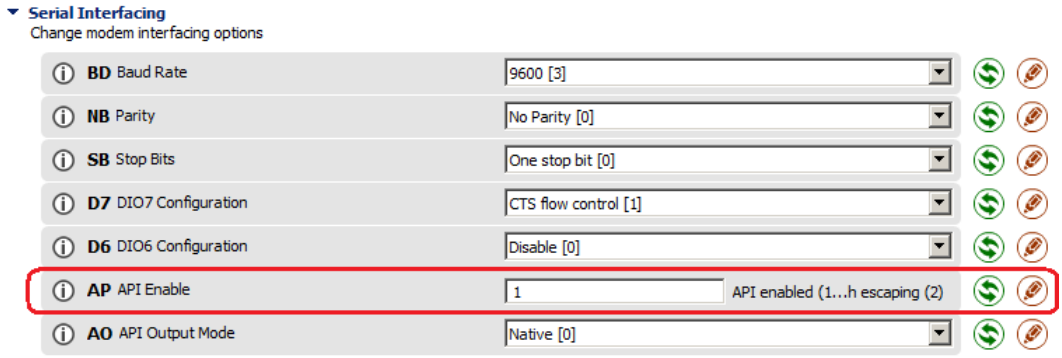

Figura 25. Configuración del comando AP en valor 1.

5.3. Formato de datos

Un aspecto clave a solucionar en este proyecto es cómo se transmitirían las mediciones y los datos provenientes de sensores y actuadores. La idea propuesta es crear una estructura similar a un marco API en la parte de RF Data. Como se

explicó en el apartado de prestaciones, lo ideal es reducir el tamaño de los mensajes a transmitir, para disminuir el consumo energético de las radios y el tráfico de la red. Por esta razón se propone una estructura como la siguiente:

Tabla 7. Estructura utilizada para la transmisión de mensajes.

Data type	Device	A-S data	
Byte 1	Byte 2	Byte 4…	…Byte n
Single Byte	Single byte	Data bytes	

En esta estructura el primer byte se refiere al tipo de datos que se recibirán, como mediciones provenientes de sensores, posición o estado de actuadores o valores de parámetros. Por ello se propone la siguiente codificación, aunque en evoluciones posteriores del sistema se podrán agregar opciones adicionales.

Tabla 8. Codificación propuesta de los data type.

Data type	Descripción
0x01	Datos de medidas de sensores.
0x02	Estado y/o posición de los actuadores.
0x03	Valor actual de un parámetro.

El siguiente Byte identifica el dispositivo (sensor, actuador u otro) del que provienen los datos. De esta manera, cada dispositivo conectado al nodo sensor deberá identificarse con un byte único para transmitir datos cuando se programe cada aplicación. Este byte servirá, también, para identificar los datos en el nodo central y almacenarlos correctamente.

Finalmente aparecen los propios datos a transmitir. Como se comentó en apartados anteriores, los datos sólo incluirán los valores numéricos, mientras que otras características como unidades se asociarán a la hora de almacenar los datos en el nodo central, a partir del identificador del dispositivo.

En Arduino los números enteros o `int` ocupan 16 bits; las variables de tipo `long`, 32 bits; y los `floats`, también 32 bits. Esto supone una limitación para el tamaño de los números con el que se puede trabajar y es un aspecto muy importante a la hora de transmitir los datos a través de la red. Teniendo en cuenta que las radios XBee en modo API requieren comunicarse a través de

bytes, se plantearon y sopesaron 2 posibles formas de pasar los datos numéricos:

- ASCII

Esta forma consiste en convertir la medición, transformando los datos numéricos a un array de caracteres o `char` los cuales se envían byte a byte por la radio. De esta forma, cualquier número independientemente de si se trata de un entero int o un número decimal float se puede convertir fácilmente.

  - -27.36 → "-27.36"
  - 1.9 →"1.9"

Cuando se envíen múltiples medidas de esta forma resulta necesario indicar la longitud de cada una, el número de bytes que ocupa, ya que cada cifra se envía separada del resto, incluida la coma decimal.

Por otro lado, en el receptor final se deben recomponer los números a partir de los caracteres. Esto requeriría utilizar un algoritmo relativamente complejo que debería convertir las cifras a un número acumulándolas, detectar la posición del punto decimal y tratar las cifras decimales acumulándolas y sumándolas. Como se comentó, este método requeriría indicar la longitud de cada número transmitido, la cantidad de bytes que ocupa, si se envían múltiples mediciones juntas, lo que complicaría aún más el algoritmo. Este inconveniente se podría solucionar limitando a un solo dato numérico por mensaje, aunque esto supondría incrementar el tráfico de la red y el consumo de energía al tener que enviar múltiples transmisiones.

El punto fuerte de este método es que permite enviar medidas muy grandes y/o pequeñas, ya que con éste los números pueden extenderse tanto como sea necesario.

- BYTE

En esta forma se plantea dividir los números a transmitir en bytes mediante operaciones `Bitwise`. De esta forma, cualquier número entero se podría transmitir utilizando sólo dos bytes, dividiéndolo en su parte alta y baja. Las variables de tipo `long` se podrían tratar de forma similar pero dividiéndola en 4 bytes.

Ejemplo:

  - -1024 →0xFC 0x00 (1111 1100 0000 0000)

- 40→ 0x00 0x28 (0000 0000 0010 1000)

La diferencia es que los números no se transmiten a través del código ASCII de cada una de sus cifras, sino en la forma binaria en la que son almacenados. El problema de esta forma es que aunque Arduino trabaja con enteros de 16 bits, los ordenadores para el almacenamiento de datos son sistemas de 32 bits. Esto hace difícil reconocer y reconstruir números negativos enviando sólo dos bytes de información. Por esta razón se plantea incluir un byte adicional de signo. De esta forma, cuando este byte tenga valor 0x01 será indicativo de un número negativo, mientras el número en sí se enviará en valor absoluto

- -1024 →0x01 0xFC 0x00 (0000 0001 0000 0100 0000 0000)
- 40→ 0x00 0x28 (0000 0000 0000 0000 0010 1000)

Las variables de tipo `long` requerirán entonces cinco bytes para ser transmitidas.

Por su parte las variables de tipo float son un poco más complejas de manipular. Dado que no se puede tener acceso para partir los 32 bits de este tipo de variables en 4 bytes, se optó por una solución alternativa. Esta solución consiste en separar este tipo de variables en dos enteros, uno para su parte entera y otro para la decimal y transmitirlos como tales. Como resultado, todo float se puede trasmitir utilizando 5 bytes: uno para el signo, dos para la parte entera y otros dos para la parte decimal.

Ejemplo:

- -27.36 → 0x01 0x00 0x1b 0x00 0x24

Aunque debido a características propias de Arduino la forma en que transmitirá el número es:

- -27.36 → 0x01 0x00 0x1b 0x8C 0xA0

La parte decimal se transforma en un entero de 16 bits lo más grande posible. En este caso el equivalente entero de la parte decimal del número es:

- 0x8C 0xA0 → 36000

Teniendo en cuenta las posibles variables con las que se podrá trabajar, se planteó un marco único de cinco bytes para transmitir cualquier número, independientemente del tipo de variable del que se trate.

Tabla 9. Estructura del formato para la transmisión de datos.

Signo	0x00
Bytes superiores o parte entera	0x00
	0x00
Bytes inferiores o parte decimal	0x00
	0x00

Un número entero se transmitirá utilizando los dos bytes superiores, mientras que los bytes inferiores se utilizarán al transmitir variables `float` o `long`. De esta mera cada número siempre ocupará lo mismo y es más fácil delimitarlo y procesarlo para recuperarlo en destino mediante operaciones nversas a las aplicadas para partirlo.

El inconveniente de este sistema es que limita el número que se puede transmitir. Un float no puede ser mayor a 65536,9999. Pero dadas las características de los dispositivos utilizados en este proyecto y otros aparatos que se suelen emplear en Arduino (básicamente en lo referido a la resolución y la precisión), no parece resultar particularmente significativa. En todo caso, se recomienda trabajar con múltiplos de unidades y manejar los datos en origen antes de transmitirlos.

Por todo lo explicado el método escogido para la transmisión de los datos es el que se acaba de explicar.

5.4. Software del nodo sensor

El software del nodo sensor debe ser capaz de recoger mediciones y de realizar actuaciones de todos los dispositivos conectados a éste. El software debe permitir transmitir a través de la radio XBee los registros, teniendo en cuenta el funcionamiento en modo API. Además de todo esto, el software debe permitir ahorrar tanta energía como sea posible poniendo el nodo en modo SLEEP, si así es necesario.

El programa del nodo se desarrolló a través del software de programación de Arduino o IDE. Aquí se podrán programar fácilmente el software de control de los dispositivos conectados al nodo pudiéndose incluir librerías para facilitar la operación.

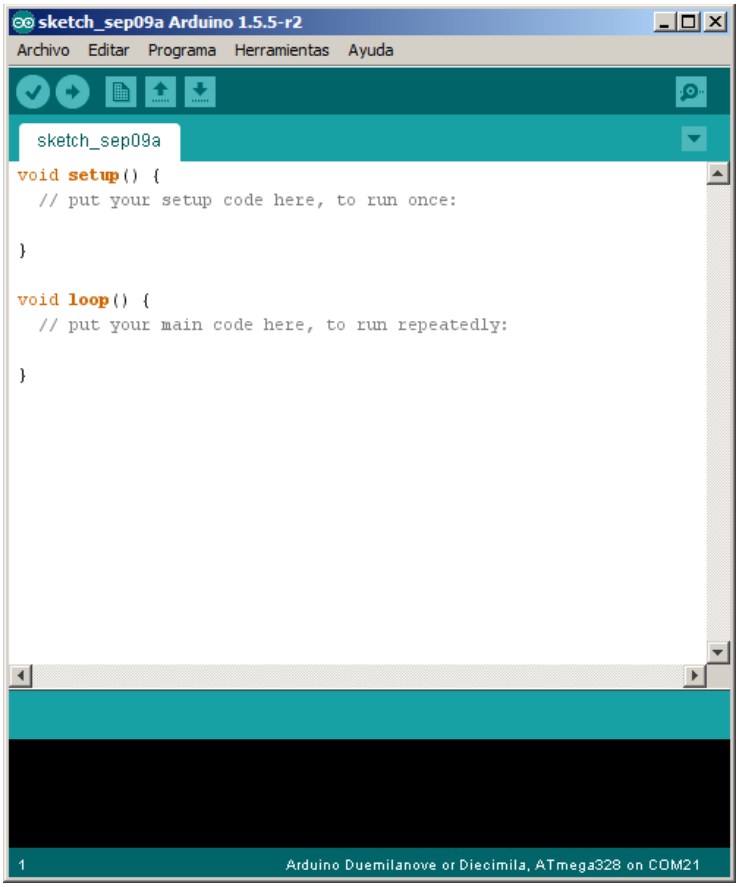

Figura 26. IDE de Arduino mostrando el Bareminimum.

Con el fin de disminuir el consumo de energía se plantearon dos esquemas de funcionamiento alternativo para el programa del nodo. El primero y el más simple mantiene el nodo encendido en todo momento, realizando mediciones de los sensores de forma cíclica con una frecuencia programable por el usuario. En este proyecto se ha optado por evitar el uso de funciones `delay` para contabilizar el tiempo ya que no permite realizar otras acciones mientras el temporizador está funcionando. Este esquema de funcionamiento permite mayor flexibilidad de funcionamiento al estar siempre encendido y pendiente a las instrucciones provenientes del nodo central.

El segundo, pensado para ahorrar energía, aprovecha las características de Ardiuno y XBee haciendo uso de modo SLEEP configurable. Para ofrecer largos periodos en estado apagado se ha optado por una configuración diferente a la convencional. Normalmente se utilizaría un temporizador de Arduino para despertar y poner a dormir el nodo, siendo éste quien enciende y apaga la radio. La desventaja de este método es que el mayor temporizador de Arduino sólo llega a los 8 segundos y es bastante complicado de programar [8]. Por su parte el módulo XBee se puede programar muy fácilmente a través de X-CTU y dispone

de un modo sleep de hasta 28 segundos, ampliables mediante comandos avanzados.

Por esta razón se optó por configurar el software de tal modo que sea el módulo XBee quien gobierne los ciclos SLEEP/WAKE UP y que sea éste quien encienda al Arduino. Una característica añadida de esto es que mediante comandos AT remotos, sólo disponibles en modo API, se puede modificar e incluso anular fácilmente el modo SLEEP y adaptar el nodo a las necesidades del usuario.

En ambos casos, y mientras el nodo esté encendido, las instrucciones recibidas por el puerto serie serán procesadas mediante la función `serialEvent()` que funciona como interrupción. Una vez iniciado el puerto serie y recibida la confirmación del módulo XBee que se encuentra conectad a la red, cada vez que se reciben datos por el puerto se ejecuta esta función que recoge todos los mensajes y ejecuta las acciones pertinentes, descifrando el marco API recibido.

Cada dispositivo conectado al nodo (sensor o actuador) deberá programarse de acuerdo a cada aplicación de tal forma que los datos a transmitir formen un array de bytes según el formato descrito arriba Los datos se enviarán al módulo XBee mediante una función especialmente creada en este proyecto para trabajar en modo API. Esta función construye todo el marco del comando *TX request* a partir de los datos y la dirección del nodo coordinador programada en Arduino y los envía a través del puerto serie al módulo XBee.

El desarrollo de las funciones de comunicación en modo API entre Arduino e XBee fue una de las partes fundamentales de este proyecto. Esta tarea fue necesaria dado que se descartó el uso de librería especializadas en comunicación API con XBee. Las razones de esto fueron por una parte la reducción del espacio ocupado en la memoria, pero también la posibilidad de aprender de primera mano el funcionamiento de los módulos XBee y desarrollar un código sencillo para el proyecto. Otra razón es la limitada información sobre el funcionamiento de las librerías existentes que, por otro lado, son complicadas de descifrar.

Como resultado de esto se pudo comprobar de primera mano que la complejidad de tener que trabajar con los marcos API no era tan justificada. Gracias al manual de XBee desarrollado por DIGI [7] y otras referencias consultadas [5], en las que se explica de forma detallada la composición y construcción de los marcos, el trabajo se limitó a tener que manipular unos simples bytes. Otra ventaja de las funciones desarrolladas en este proyecto es que permiten la transmisión de datos con el formato diseñado y explicado arriba.

Seguidamente se muestra el código de una de las funciones más importantes de todo el proyecto. Se trata de la función de transmisión de datos tx_request(), que dado un array de bytes en el que se incluyen las mediciones de acuerdo al formato de datos desarrollado, crea el marco y lo envía el módulo XBee para su transmisión. El lector podrá comparar la terminología con la tabla 6 y seguir el funcionamiento más cómodamente.

Tabla 10 . Código de la función tx_request().

```
//Dirección de destino. 64 bits + 16 bits del coordinador
uint8_t
addr[10]={0x00,0x13,0xA2,0x00,0x40,0xBA,0xBC,0x78,0x00,0x00};

void tx_request(uint8_t *rfdata, int datasize){
  //1xframetype, 1xframeid, 8x64bitaddr, 2x16bitaddr,
1xbroadcastradio, 1xoptions
  int packet_lenght=14+payloadsize;

  uint8_t frame_type=0x10;
  uint8_t frame_ID=0x01;
  uint8_t broad_radius=0x00;
  uint8_t options=0x00;
  uint8_t checksum=0xFF;

  //cálculo del checksum
  int sum=frame_type+frame_ID+broad_radius+options;
  for(uint8_t i=0; i<10; i++){sum+=addr[i];}
  for(uint8_t i=0; i<datasize;i++){sum+=rfdata[i];}
  checksum-=lowByte(sum);

  //transmisión del marco
  Serial.write(0x7E);
  Serial.write(highByte(packet_lenght));
  Serial.write(lowByte(packet_lenght));
  Serial.write(frame_type);
  Serial.write(frame_ID);
  for(uint8_t i=0; i<10;i++)Serial.write(addr[i]);
  Serial.write(broad_radius);
  Serial.write(options);
  for(uint8_t i=0; i<datasize;i++)Serial.write(rfdata[i]);
  Serial.write(checksum);
}
```

Como resultado del trabajo desarrollado en este apartado, el programa base que se empleará en el Arduino del nodo sensor, encargándose de las comunicaciones y los ciclos de SLEEP, ocupa sólo un 12% de la memoria flash y utiliza un 9% de la memoria RAM. Esto permite dejar bastante espacio libre en el nodo para programar el MCU para las aplicaciones específicas que se deseen.

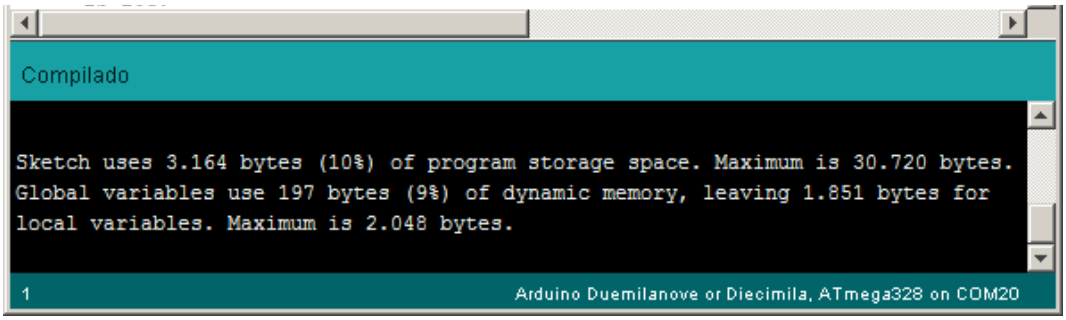

Figura 27. Ventana de estado de Arduino mostrando el resultado de la compilación del programa del nodo sensor.

A continuación pueden verse los dos diagramas de flujo de los programas propuestos para los el nodo sensor, mientras que en el anexo correspondiente puede verse el código programado para Arduino con comentarios para su interpretación.

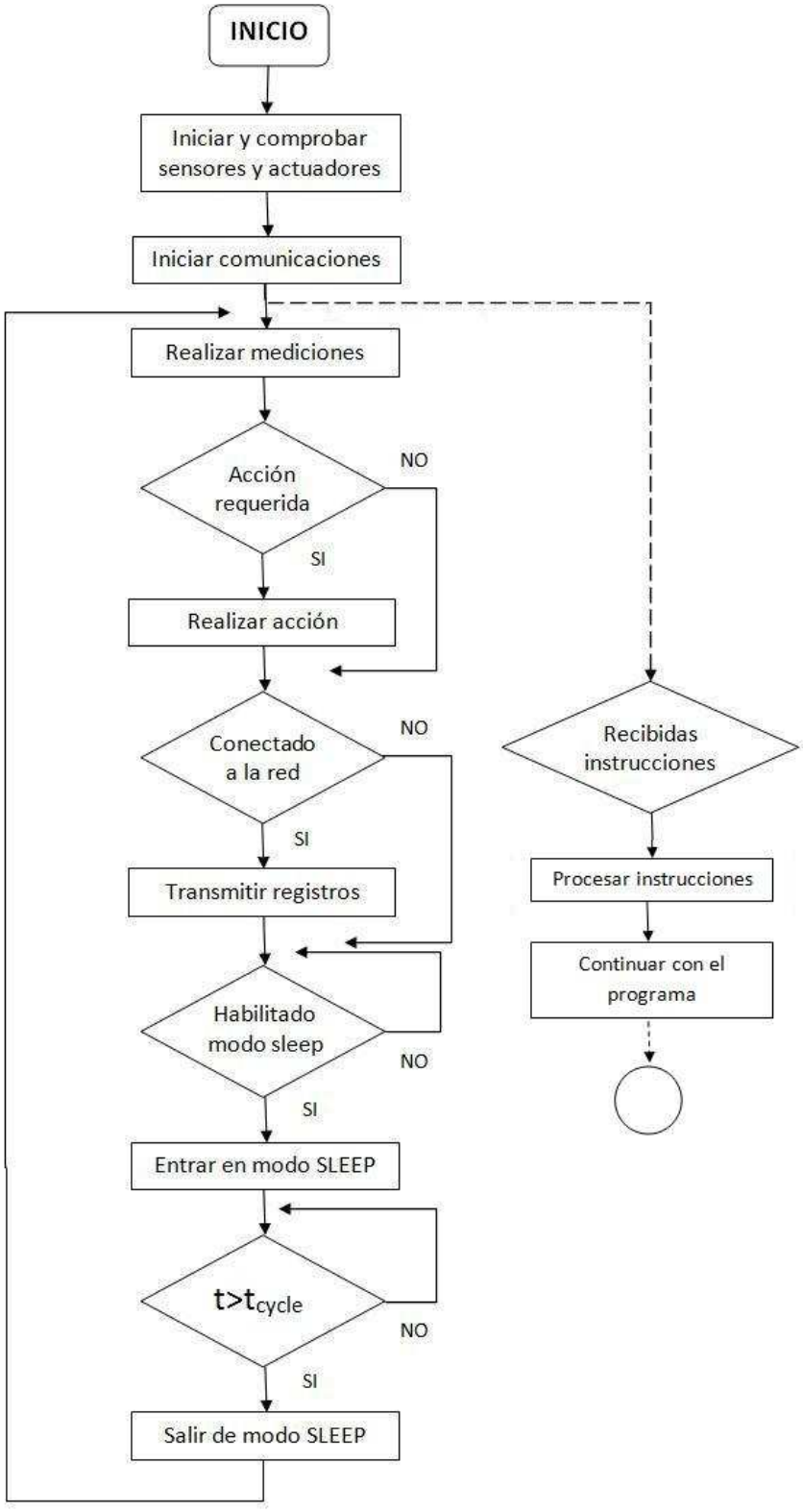

Figura 28. Diagrama de funcionamiento del software del nodo con modo SLEEP habilitado.

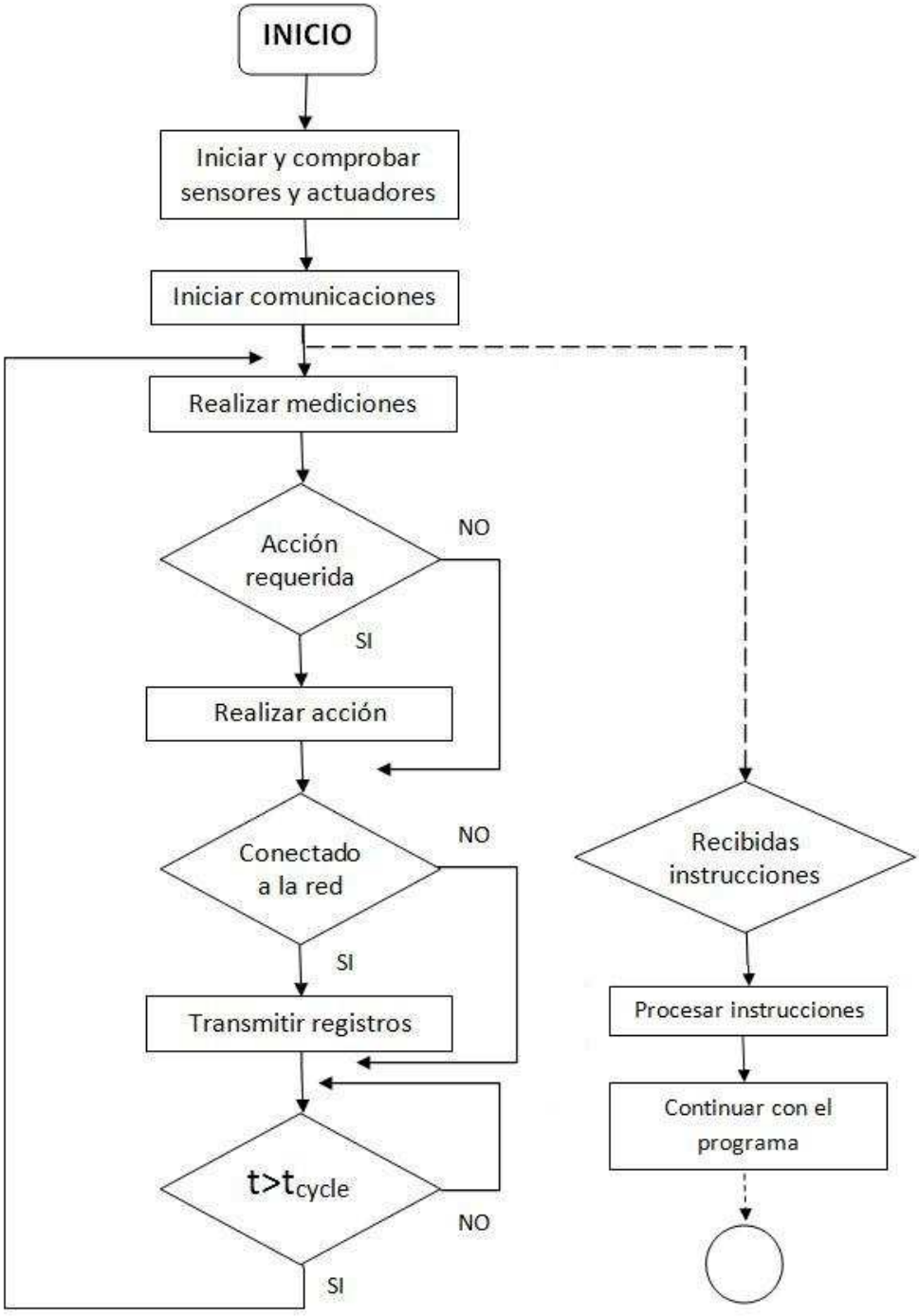

Figura 29. Diagrama de funcionamiento del software del nodo sin modo SLEEP.

5.5. Software de procesado de datos

El programa desarrollado en Processing para procesar los datos recibidos desde la red debía cumplir una serie de características especificadas en el capítulo de prestaciones. Se programó con el objetivo de facilitar cuanto fuera posible su uso al usuario, creando un entorno en el que se pudiera administrar fácilmente la red, sin tener que complicarse modificando manualmente archivos del programa para hacerlo funcionar. Aunque esta característica complicó más la programación, el resultado es un programa más visual y flexible que le otorga al usuario mayor control e información.

5.5.1. Archivos y aspectos gráficos

La principal característica de este programa es el uso de objetos tipo `Table` para procesar los datos. Como su nombre indica se trata de un tabla en la cual se pueden escribir y leer datos de forma sencilla y que pueden ser creadas y modificadas de forma dinámica dentro del programa. Las tablas también se pueden generar cargando desde un archivo en el disco duro, y se pueden guardar en archivos, salvando los datos.

Por defecto en Processing las tablas utilizan archivos con extensión *.CSV (Comma Separeted Values), se trata de un archivo de texto plano sencillo en el que se describe una tabla. En estos, las columnas de una tabla se marcan utilizando comas para separar los valores, como su nombre indica, mientras que las líneas del texto indican el salto de filas. Son archivos que pueden ser creados y editados fácilmente mediante un editor de texto plano como el Bloc de notas de Windows, pero que tienen mucho potencial ya que pueden ser abiertos por programas de hojas de cálculo como Microsoft Excel. Esta característica permite que los datos de los sensores puedan estudiarse al detalle de forma fácil y cómoda.

En el programa desarrollado las tablas se utilizan no sólo para guardar los datos de los sensores sino también para describir la propia red y todos los dispositivos. Cada nodo que integra la red se encuentra guardado en un archivo de nombre `network.csv` que se encuentra en el directorio del programa. En este archivo se incluye la dirección de la radio XBee (sólo la parte baja de la dirección de 64 bits), el nombre asignado por el usuario al nodo, el rol de la radio (coordinador, router o end device) y la posición en la que será dibujado en el programa.

De la misma forma, cada dispositivo conectado a un nodo de la red, ya sea sensor o actuador, se almacena en un archivo de nombre `devices.csv` en la carpeta del nodo en el que se incluye la ID del dispositivo, según se explicó en el aparato de formato de datos, el nombre asignado por usuario, el tipo y la posición ara ser dibujado en pantalla.

Finalmente, cada parámetro del dispositivo del que se reciban datos se encuentra descrito en archivo específico dentro de la carpeta del nodo cuyo nombre es la ID del dispositivo. En este archivo de describe el nombre del parámetro, el tipo de datos esperado, la unidad de medida de los datos, los valores límites y la posición por defecto, si existe.

Estos archivos se utilizan no sólo para hacer funcionar la red sino también para dibujar el entorno gráfico de la aplicación. Gráficamente el programa funciona a tres niveles: el nivel de red, el de nodo y el de dispositivo. En el primero se muestran todos los nodos de la red, dibujando una etiqueta para cada uno según los datos del archivo `network.csv` .

Figura 30. Ejemplo de etiquetas de los tres tipos de nodo de la red.

Al clicar sobre el botón circular de un nodo, el programa pasa al nivel del nodo cargando el archivo `devices.csv` del seleccionado y dibujando todos los dispositivos conectados.

Figura 31. Ejemplo de etiquetas para los dispositivos de un nodo.

De forma similar, al clicar sobre el botón de un dispositivo el programa pasa al nivel de dispositivo, cargando en este caso no sólo los parámetros disponibles sino también los datos de mediciones almacenados para mostrarlos en pantalla. En este menú se puede cambiar de parámetro activo seleccionando el que se desee estudiar de la lista disponible. Cuando se selecciona un parámetro el programa carga y grafica los datos disponibles para este. También existe la

posibilidad de graficar todos los datos disponibles para este parámetro o sólo las últimas mediciones recibidas. El número de puntos graficados se puede aumentar mediante los botones +50 y -50 que incrementan o disminuyen el número de puntos mostrados, desde un mínimo de 50 hasta 500.

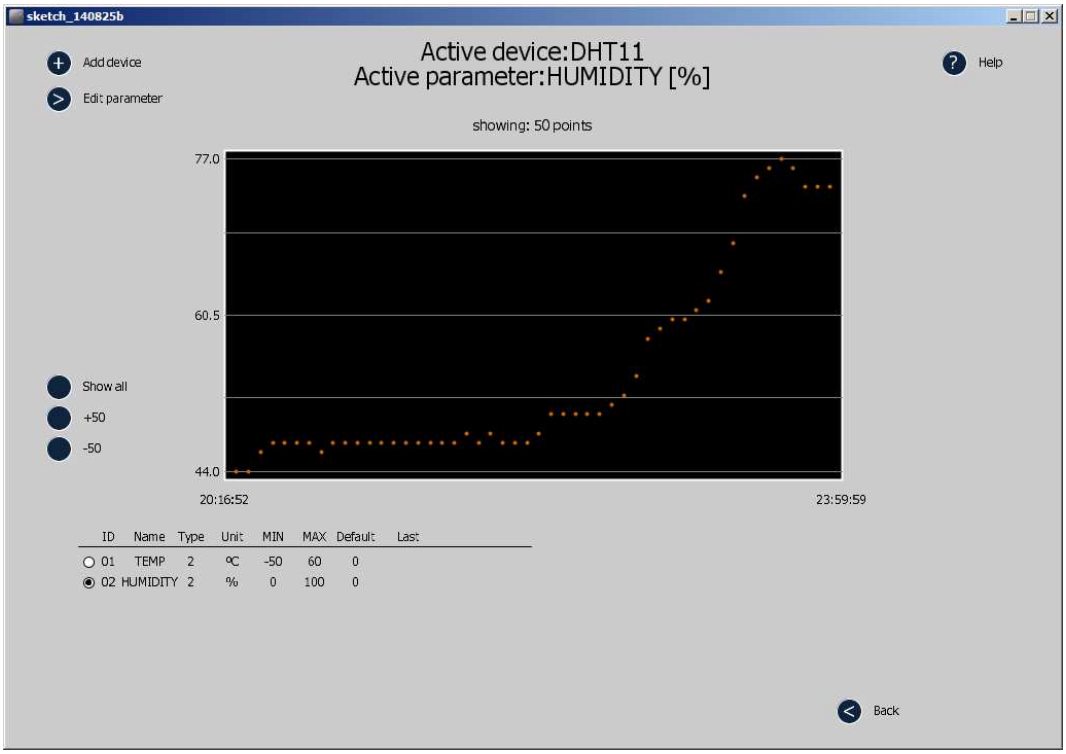

Figura 32. Programa funcionando en el nivel de dispositivo.

5.5.2. Comandos de actuación

La otra característica de la aplicación en este nivel es la posibilidad de configurar y enviar comandos de actuación al nodo sensor. De forma sencilla y cuando el dispositivo es un actuador, se habilita una ventana que permite ajustar el valor de los parámetros del actuador y enviar el comando de actuación. El programa utiliza las variables de nodo, dispositivo y parámetro, además del valor ajustado y formateado, para construir el marco API y enviarlo al nodo, donde se descifrará la instrucción y aplicará el comando, devolviendo la posición final para registrar su estado.

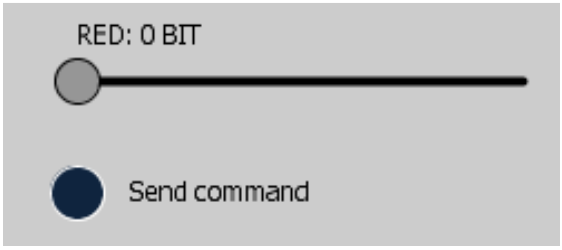

Figura 33. Ejemplo de ajuste de actuación para la intensidad de un led rojo.

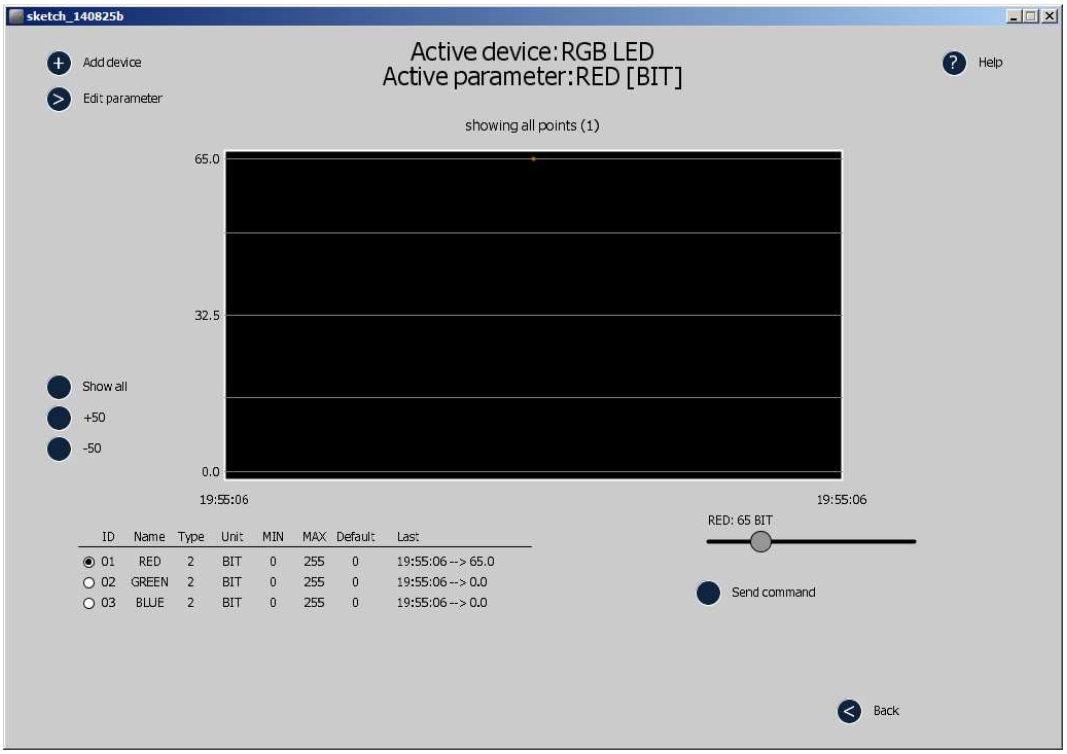

Figura 34. Programa funcionando a nivel de dispositivo con un led RGB como actuador configurable.

Un inconveniente del programa desarrollado es que sólo se pueden ejecutar comandos de actuación un parámetro a la vez. Por otro lado, y como se explicó, los mensajes de datos de un dispositivo se envían todos juntos desde ARduino, diferenciándose unos de otros por su posición en el mensaje. El resultado es que las aplicaciones desarrolladas, tanto en Processing como en Arduino, no tratan los parámetros de forma individualizada sino como componentes de un dispositivo.

En consecuencia fue necesario desarrollar una estructura que permitiera determinar qué parámetro de un actuador se desea modificar, de acuerdo al comando enviado, mientras que el resto deberá ser ignorado.

La solución fue utilizar el primer byte de un dato como señalador. De acuerdo al formato de datos planteado de cinco bytes, el primero corresponde al signo del número, siendo este 0x00 cuando el número es positivo y 0x01 cuando es negativo. Ahora se plantea que este byte tome el valor 0xFF cuando el parámetro deba ser ignorado al transmitir un comando de actuación. De esta forma, cuando la aplicación de Arduino en el nodo sensor lea este byte entiende que no debe modificar el parámetro del actuador y que debe saltar al siguiente parámetro.

Adicionalmente esta estructura permite que los parámetros no modificados o comandados ocupen sólo un byte (0xFF) en el mensaje, mientras que el parámetro a comandar ocupará los cinco bytes, con el primero señalando el signo correspondiente.

Esta es la estructura que se programó en Processing y con la que se recibirán los mensajes en Arduino. Las funciones específicas para comandar los actuadores que se conecten al nodo sensor deberán programarse teniendo en cuenta esta estructura y adaptándose a cada caso.

A modo de ejemplo se plantea el siguiente caso:

5.5.2.1. Ejemplo

El actuador a en cuestión es un diodo led RGB con tres parámetros comandables, uno por cada color. Como puede verse en la figura 34, la aplicación de Processing tiene configurados los tres parámetros y está lista para enviar el mensaje de actuación.

Suponiendo que sea el segundo de los tres parámetros el que se comanda, la estructura del mensaje a enviar al nodo sensor es la siguiente:

Tabla 11. Estructura del mensaje de actuación para un led RGB.

Data type		0x02
Device		0x05
A/S data (comando de actuación)	1º param. (ignorar)	0xFF
	2º param. (actuar)	0x00
		0x00
		0x1B
		0x00

		0x00
	3º param. (ignorar)	0xFF

Cuando Arduino recibe este mensaje identificando un comando de actuación (Data type), recompone el comando y ejecuta la función correspondiente para el dispositivo señalado (0x05). A continuación se muestra una parte de código encargado de aplicar el comando sobre el led.

Tabla 12. Ejemplo de una función de comando de un led RGB.

```
uint8_t bright[3]={0,0,0};

void light (uint8_t R, uint8_t G, uint8_t B){
  R=R^0xFF;
  G=G^0xFF;
  B=B^0xFF;

  analogWrite(RED, R);
  analogWrite(GREEN, G);
  analogWrite(BLUE, B);

}

void LEDstate(){
  uint8_t led[17];
  led[0]=0x01;
  led[1]=actuatorLED;
  uint8_t pos=2;
  ftohex(float(bright[0]), value);
  for(uint8_t i=0; i<sizeof(value); i++)led[pos+i]=value[i];
  pos+=5;
  ftohex(float(bright[1]), value);
  for(uint8_t i=0; i<sizeof(value); i++)led[pos+i]=value[i];
  pos+=5;
  ftohex(float(bright[2]), value);
  for(uint8_t i=0; i<sizeof(value); i++)led[pos+i]=value[i];
  tx_request(led, sizeof(led));
  }

.
.
.
 case 0x02:  if (rfdata[1]==0x05){
                      uint8_t pos=2;
                      for(uint8_t j=0; j<3; j++){
                        //si el primer byte es 0xFF ignorar
parámetro
                        if(rfdata[pos]==0xFF)pos++;
                        //de lo contrario procesar valor
                        else{
                          bright[j]= rfdata[pos+2];
                          pos+=5;
                          }
                      }
                        //ejecutar funcion de actuación del led
```

```
                        light(bright[0], bright[1], bright[2]);
                          //enviar resultados de la actuación
                        LEDstate();
                }
                break;
.
.
.
```

Puede verse que el código es sencillo y ha sido desarrollado específicamente para este actuador, aprovechando las características del mensaje para componer los valores de cada parámetro antes de ejecutar la función de actuación.

5.5.3. Edición de la red

Como se explicó, el programa se diseñó con el objetivo de facilitar el uso de la red al usuario. Por ello, el programa dispone de un modo de edición que permite añadir y editar nodos y dispositivos de la red. Mediante el uso del cuadro de edición se puede editar la red de forma fácil y cómoda mientras el programa se encarga de crear los archivos que sean necesarios.

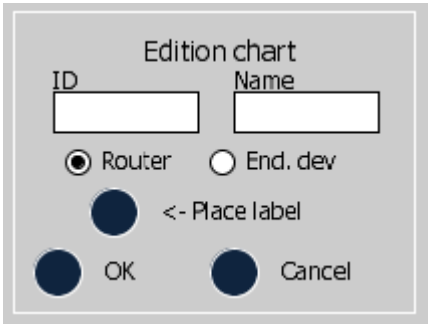

Figura 35. Cuadro de edición del programa en el nivel de red.

Mientras el modo de edición esté activo, se almacena en una variable auxiliar los parámetros modificados, de tal forma que al confirmar los cambios se ejecuta la función que agrega el elemento a la red, modifica sus características existentes, o lo elimina si se borra la ID de éste durante la edición.

5.5.4. Comunicaciones

Paralelamente a los aspectos gráficos, el programa debe procesar los datos recibidos por la red a través del puerto USB al que el nodo coordinador esté conectado. Por ello, el programa utiliza la función `serialEvent()`, la misma función que se utiliza en los nodos para recibir instrucciones. Aquí la función pasa los datos recibidos a otra que se encarga de reconstruirlos siguiendo el formato planteado y guarda las mediciones en el archivo de datos del dispositivo. Además

de esto, si el programa se encuentra en el nivel del dispositivo y se reciben datos de éste, éste ejecuta las funciones necesarias para incluir el nuevo punto en el gráfico. La función `rx_packet (byte rfdata[])` se encarga de esta tarea. A continuación se muestra parte del código de la función, que puede verse completa, junto al resto del código desarrollado, en el anexo correspondiente. El fragmento que se muestra se encarga de reconstruir las mediciones recibidas de acuerdo al formato planteado y de guardarlas en el archivo correspondiente.

Tabla 13. Fragmento de la función rx_packet.

```
  if (type==0x01){
    //al paquete recibido le restamos la dirección, las opciones,
el checksum, el dispositivo y el tipo de dato y nos quedan las
mediciones
    int messurements=(packet_length-14)/5;
    float [] values=new float[messurements];
    for(int i=0; i<messurements; i++){
      int position=13+5*i;
      int sign=int(rfdata[position])*(-2)+1;
      int
entero=int(rfdata[position+1])*256+int(rfdata[position+2]);
      float
decimal=float(rfdata[position+3])*256+int(rfdata[position+4]);
      while (decimal>1) decimal=decimal/10;
      values[i]=sign*(entero+decimal);
    }
    //guardar las mediciones

rxdata=loadTable("data/"+fecha+"/"+node+"/"+hex(device,2)+".csv",
"header");
    TableRow newRow = rxdata.addRow();
    newRow.setString(0, hora);
    for (int i=0; i<messurements; i++){
    newRow.setFloat(i+1, values[i]);
    }
    saveTable(rxdata,
"data/"+fecha+"/"+node+"/"+hex(device,2)+".csv");

.
.
.
```

Los datos recibidos se almacenan en un archivo con la ID del dispositivo al que pertenecen, en la carpeta del nodo correspondiente, y dentro de una carpeta con la fecha del día. Cada vez que se inicia el programa se crean los archivos de todos los dispositivos de la red en esta carpeta para que, cuando se reciban nuevos datos, el programa pueda abrir el archivo, incluir las mediciones y volver a guardarlo. Igualmente, cada vez que la fecha cambia se vuelven a crear los nuevos directorios y archivos para almacenar datos. La función `Filesgen()` es la encargada de realizar esta tarea una vez al día.

Una característica de este sistema es que dado que los datos provenientes de los nodos se identifican sólo por el dispositivo, cuando uno de estos envía datos de múltiples parámetros, el orden en que se compone el mensaje es fundamental. El mensaje de datos del sensor en Arduino se debe componer con los parámetros en el mismo orden en que aparecen en la tabla del programa, en el nivel del dispositivo. A modo de ejemplo, en la figura 32 puede verse que la lista de parámetros muestra primero la temperatura y luego la humedad, de esta manera para que los datos se procesen de forma correcta tienen que recibirse en el mensaje de mediciones en el mismo orden.

El programa funciona principalmente mostrando por pantalla el menú actual, pero también está pendiente del puerto USB, del ratón y el teclado, que son los elementos de interacción del usuario. Puede verse el código completo de la aplicación desarrollada en Processing con comentarios para facilitar su interpretación en el anexo correspondiente. A continuación se muestran los diagramas de flujo del programa para comprender mejor su funcionamiento.

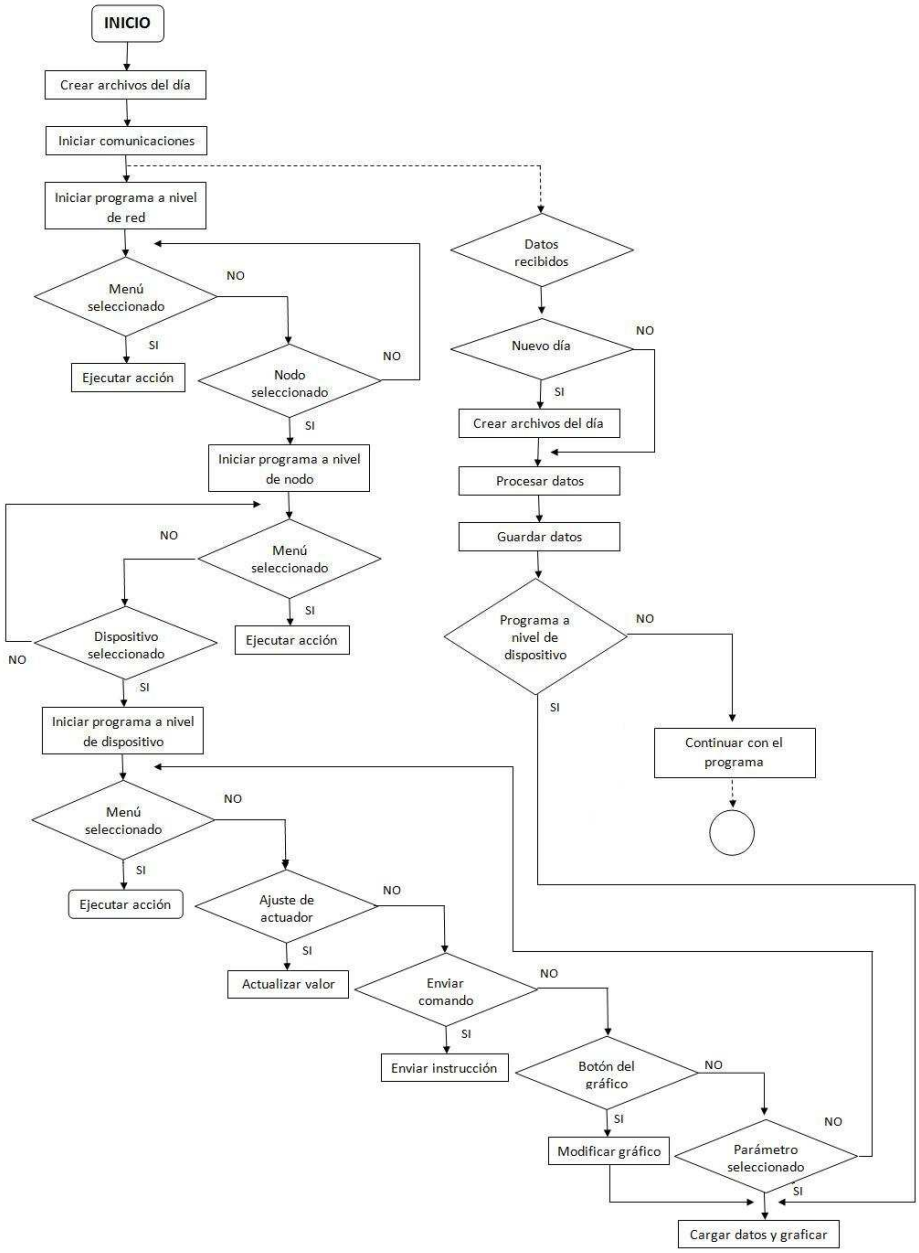

Figura 36. Diagrama de flujo principal de la aplicación.

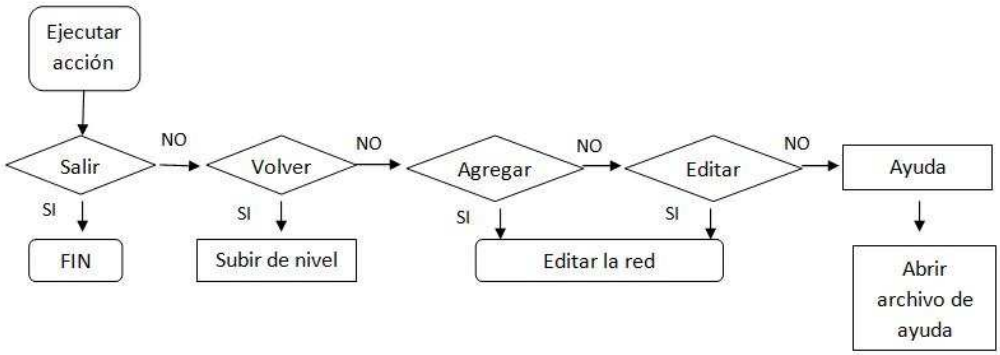

Figura 37. Diagrama de flujo de los botones generales de la aplicación.

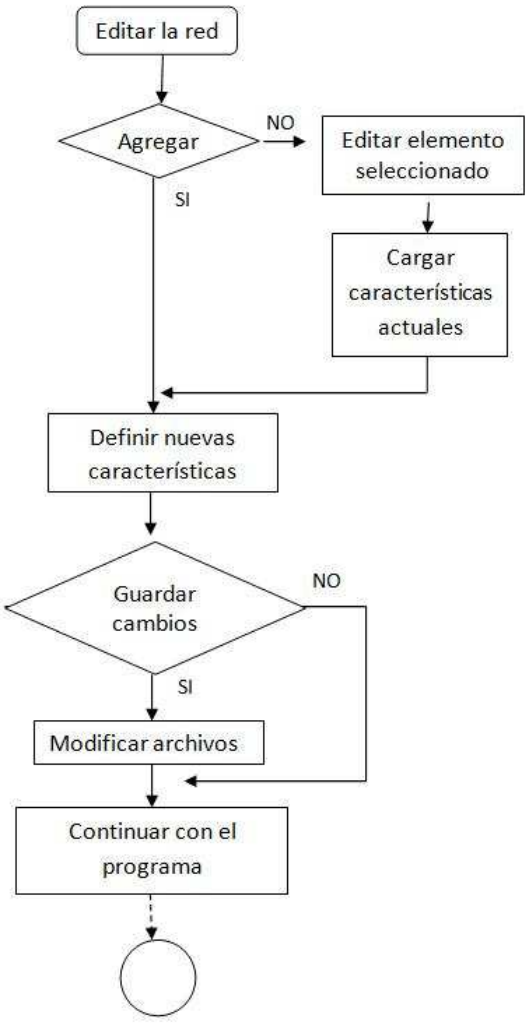

Figura 38. Diagrama de flujo para el cuadro de edición de la red.

6. Implementación y pruebas de funcionamiento

A lo largo del proyecto se realizaron varias pruebas de funcionamiento, la mayoría de las cuales iban dirigidas a probar conceptos o partes del código implementado. Pero tras terminar de programar se decidió realizar una serie de ensayos para comprobar el funcionamiento del conjunto. A continuación se resumen las más significativas de las pruebas realizadas.

6.1. Prueba potenciómetro

Objetivo

El objetivo de esta prueba es comprobar el funcionamiento el programa de captura de mediciones del nodo sensor, y el sistema de procesado de datos desarrollado en Processing.

Material utilizado

Para esta prueba se utilizaron dos nodos para conformar la red, uno como el coordinador de la red y el otro como nodo sensor. En el nodo sensor se conecta un potenciómetro que funcionará como sensor. El nodo sensor será alimentado mediante el adaptador o transformador.

- 2x placas Arduinoo, una sin el MCU.
- 2x Wireless SD shields.
- 2x radios XBee.
- 1x Potenciómetro de 4,7 kΩ
- Adaptador AC/DC
- Cable USB
- Placa de prototipado
- Cables de conexión

Metodología

El módulo XBee del nodo sensor se configuró como router API con el modo SLEEP deshabilitado, mientras que el nodo central utilizaró la radio XBee como coordinador en modo API. Ambos en modo API 1 y formando parte de la red con ID 2014. Se conectarán por defecto el uno al otro.

Tabla 14. Resumen de la configuración de los parámetros de las radios.

Parámetro	Nodo central	Nodo sensor
Perfil	Coordinador API	Router API
ID	2014	2014
AP	1	1
DH	00 13 A2 00	00 13 A2 00
DL	40 BA BC 33	40 BA BC 78
SM	-	0

El programa del nodo sensor se configuró utilizando el puerto analógico 0 de Arduino para el potenciómetro, empleando una frecuencia de muestreo de 2 segundos. El conmutador de la shield se mantuvo en la posición MICRO. Para el potenciómetro se programó una función para realizar funciones y formatear los datos antes de enviarlos.

Tabla 15. Función para montar las mediciones del potenciómetro.

```
void pot_messurement(){
  int pot=analogRead(0);
  //ID del dispositivo
  sensor[0]=0x01;
  //tipo de mensaje
  sensor[1]=0x01;
  //signo
  sensor[2]=0;
  //parte entera
  sensor[3]=highByte(pot);
  sensor[4]=lowByte(pot);
  //parte decimal
  sensor[5]=0x00;
  sensor[6]=0x00;
  tx_request(sensor,sizeof(sensor));
}
```

En el programa de procesado de datos se configuraron las direcciones de los nodos, se añadió el potenciómetro como sensor y se ajustó el parámetro de medida en bits.

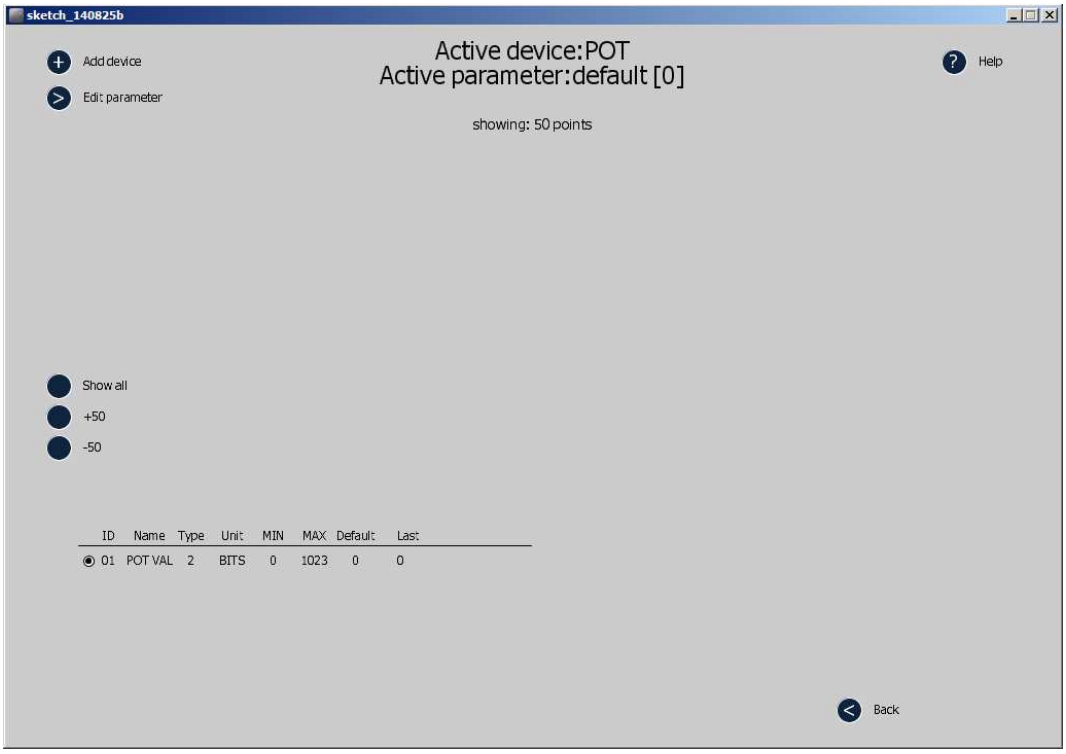

Figura 39. Configuración del parámetro utilizado en la prueba

Resultados

Se ejecutó la aplicación durante unos dos minutos en los que se recogieron poco más de 50 mediciones.

Tabla 16. Mediciones obtenidas en la prueba.

time	1	time	1	time	1
del	0	21:33:19	752	21:33:59	236
21:32:27	580	21:33:21	779	21:34:01	145
21:32:29	580	21:33:23	786	21:34:03	65
21:32:31	580	21:33:25	809	21:34:05	0
21:32:33	580	21:33:27	834	21:34:07	1
21:32:42	581	21:33:29	858	21:34:09	1
21:32:51	580	21:33:31	932	21:34:11	1
21:32:53	580	21:33:33	970	21:34:13	1
21:32:55	580	21:33:35	1020	21:34:15	1
21:32:57	582	21:33:37	1022	21:34:17	1
21:32:59	580	21:33:39	1023	21:34:19	1
21:33:01	581	21:33:41	1023	21:34:21	1
21:33:03	581	21:33:43	1023	21:34:23	1
21:33:05	580	21:33:45	1023	21:34:25	1
21:33:07	580	21:33:47	1023	21:34:27	1

21:33:09	583	21:33:49	899	21:34:29	1
21:33:11	581	21:33:51	733	21:34:31	1
21:33:13	684	21:33:53	627	21:34:33	1
21:33:15	722	21:33:55	462		
21:33:17	737	21:33:57	362		

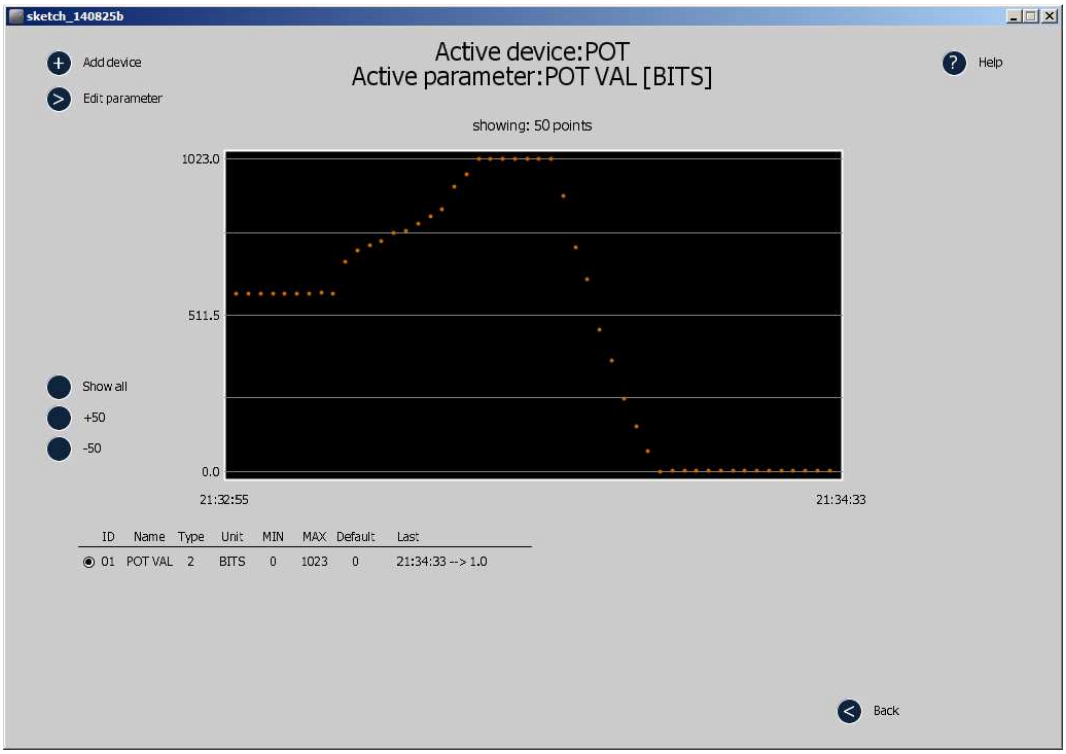

Figura 40. Mediciones obtenidas graficadas en el programa.

Análisis de los resultados

Con esta prueba se comprobaron varios aspectos de la programación del sistema. Por un lado, las funciones para construir y descifrar los marcos API demostraron ser correctas y sólo algunas pequeñas modificaciones fueron necesarias. Por el otro, el programa de procesado de datos requirió de algunas correcciones para hacer funcionar correctamente el modo de edición y algunos detalles gráficos, como en el gráfico de mediciones. En general el resultado de esta prueba fue satisfactorio.

6.2. Prueba larga exposición

Objetivo

El objetivo de esta prueba era estudiar el funcionamiento de un sensor con múltiples parámetros durante un largo periodo de tiempo. Se estudió también el comportamiento del nodo sensor con el modo SLEEP habilitado.

Material utilizado

A diferencia del ensayo anterior, aquí se utilizó un sensor digital DHT-11 que mide temperatura y humedad. Además, el nodo sensor estaba alimentado por las pilas recargables. Por lo demás, los materiales son los mismos que en el ensayo del potenciómetro.

- 2x placas Arduinoo, una sin el MCU.
- 2x Wireless SD shields.
- 2x Radios XBee.
- 1x Sensor DHT-11
- 1x Pack de pilas recargables AA
- 1x Cable USB
- Cables de conexión

Metodología

En esta prueba la radio del nodo sensor se configurará ajustada a la red del coordinador. También se configurará el modo SLEEP en de forma cíclica para que la radio tenga un ciclo de unos 5 minutos: 5 segundos encendida y el resto, apagada. La radio del nodo central estará configurada de la misma forma que en la prueba anterior.

Tabla 17. Resumen de la configuración de los parámetros de las radios.

Parámetro	**Nodo central**	**Nodo sensor**
Perfil	Coordinador API	Router API
ID	2014	2014
AP	1	1
DH	00 13 A2 00	00 13 A2 00

DL	40 BA BC 69	40 BA BC 69
SM	-	4
SN	-	0A
SP	-	0A F0
ST	-	13 88
SO	-	2

El programa del Arduino se configuró con el modo SLEEP habilitado y se empleó una librería específica para controlar el sensor DHT-11. Además se programó una función para componer las mediciones de los parámetros del sensor y se probó una función para formatear valores numéricos de forma general.

Tabla 18. Código de las dos funciones empleadas en esta prueba.

```
void DHT_messurement(){
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  uint8_t THM[12];
  //medida
  THM[0]=0x01;
  //dispositivo
  THM[1]=0x01;
  ftohex(t, value);
  for(uint8_t i=0; i<sizeof(value); i++)THM[i+2]=value[i];
  ftohex(h, value);
  for(uint8_t i=0; i<sizeof(value); i++)THM[i+7]=value[i];
  tx_request(THM, sizeof(THM));
}

void ftohex (float number, uint8_t result [5]){
  uint16_t aux;
  if(number<0){
    result[0]=0x01;
    number=abs(number);//eliminar el signo
  }
  else result[0]=0x00;
  aux=number;//extraemos parte entera
  result[1]=highByte(aux);
  result[2]=lowByte(aux);
  //extraer la parte decimal
  number=number-aux;
  //A será inferior al número máximo para 16 bits
  //tambien el decimal debe ser mayor a 0, sino se
quedaria en el bucle tener en cuenta los bits!!!
  while(number<6553.6 && number>0)number=number*10;
  aux=number;
  result[3]=highByte(aux);
  result[4]=lowByte(aux);
 }
```

El programa de procesado de datos se inició y se configuró para agregar el nodo, el dispositivo sensor y sus parámetros antes de iniciar la prueba.

Resultados

La prueba se desarrolló durante más de 15 horas con el nodo a la intemperie para medir las condiciones ambientales. En total se registraron 203 mediciones de cada parámetro que permitieron estudiar la evolución de las condiciones climáticas durante gran parte del día.

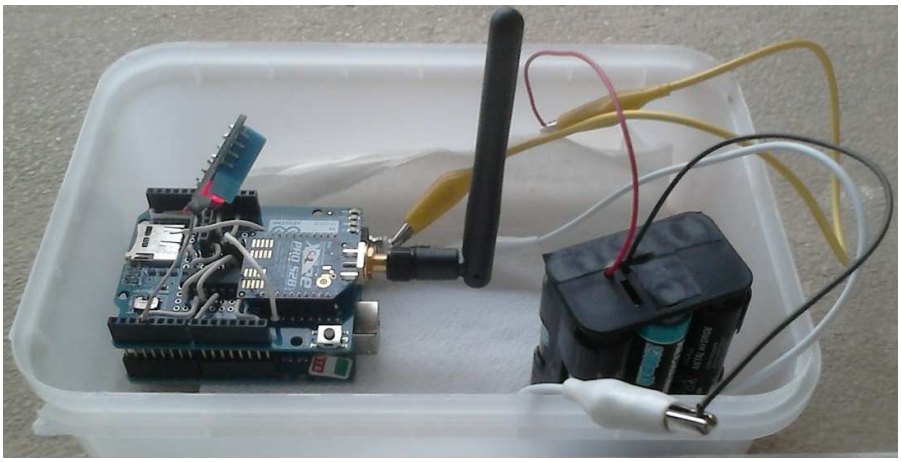

Figura 41. Nodo sensor durante la prueba.

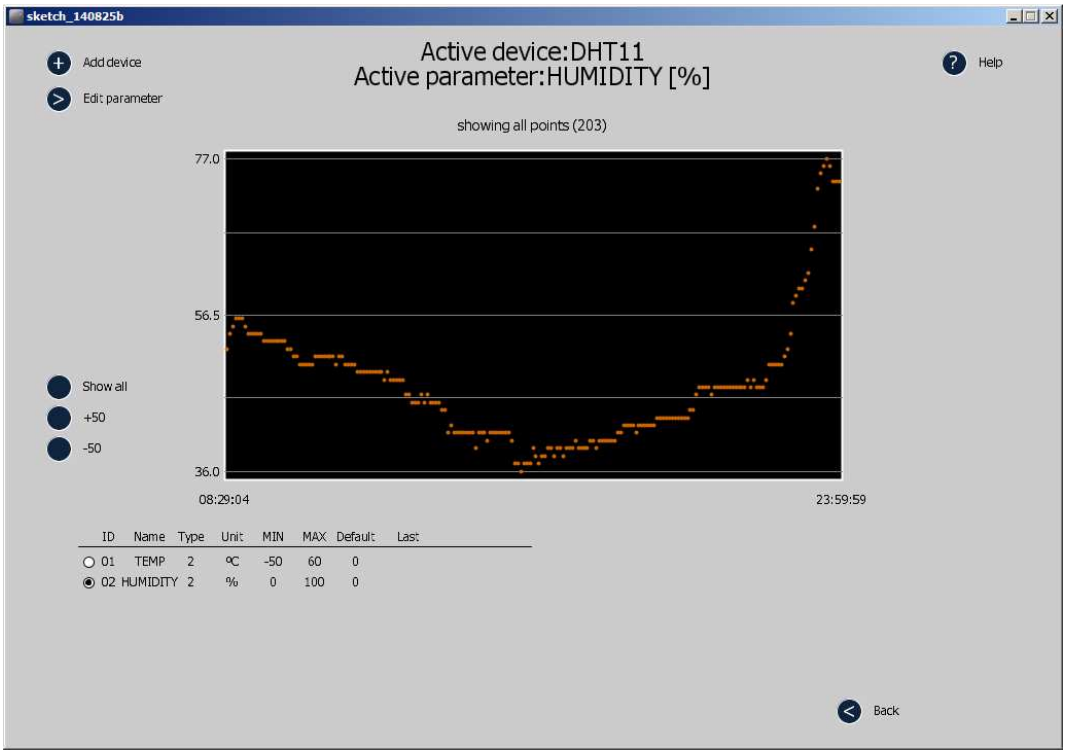

Figura 42. Datos recogidos para la humedad durante la prueba.

En el anexo de la prueba de larga exposición pueden verse los datos completos de las mediciones obtenidas durante esta prueba.

Análisis de los resultados

Con esta prueba se pudo comprobar el funcionamiento del modo sensor con el modo SLEEP habilitado, además de la función `ftohex()` que será fundamental en aplicaciones futuras para formatear los datos. En esta prueba también se pudo comprobar el completo funcionamiento de las funciones relacionadas con el gráfico de los puntos, sólo necesitando pequeñas correcciones para ajustarlo y el comportamiento del programa durante largos periodos. En general el resultado de la prueba ha sido satisfactorio.

6.3. Prueba de actuación

Objetivo

El objetivo de esta prueba es estudiar las funciones empleadas para transmitir ordenes de actuación desde el nodo central al nodo sensor.

Material utilizado

El material utilizado es el mismo que en la primera prueba, a excepción de que en este caso se utilizó un diodo led RGB como actuador para variar su intensidad.

- 2x placas Arduinoo, una sin el MCU.
- 2x Wireless SD shields.
- 2x radios XBee.
- 1x Diodo led RGB
- 1x Resistencia 330 Ω
- Adaptador AC/DC
- Cable USB
- Placa de prototipado
- Cables de conexión

Metodología

La configuración de las radios es la misma que en la primera prueba. El programa de Arduino incluía en este caso funciones para descifrar las instrucciones recibidas, funciones para aplicar los comandos al actuador y otra para enviar los mensajes de confirmación.

Tabla 19. Resumen de la configuración de los parámetros de las radios.

Parámetro	Nodo central	Nodo sensor
Perfil	Coordinador API	Router API
ID	2014	2014
AP	1	1
DH	00 13 A2 00	00 13 A2 00
DL	40 BA BC 33	40 BA BC 78
SM	-	0

En el programa de procesado de datos se configuró el nodo y el actuador con sus parámetros. De esta forma se habilitaba el modo de comando y se aplicaron numerosos comandos de actuación en los diferentes parámetros.

Tabla 20. Funciones de Arduino para descifrar las instrucciones, aplicar comandos al led RGB y para enviar confirmación.

```
void rx_packet(int packet_size){
  uint8_t payload[packet_size-12];
  uint8_t check=0x90;
  for(uint8_t i=0;i<11;i++)check+=Serial.read();
  for(uint8_t i=0;i<sizeof(payload);i++){
    payload[i]=Serial.read();
    check+=payload[i];
  }
  check+=Serial.read();//sumar checksum

  if(check==0xFF){
    switch (payload[0]){
      case 0x01:  //messurement();
                  break;
      case 0x02:  if (payload[1]==actuatorLED){
                      uint8_t pos=2;
                      for(uint8_t j=0; j<3; j++){
                        if(payload[pos]==0xFF){
                          pos++;
                        }
                        else{
                          bright[j]=payload[pos+2];
                          pos+=5;
                          }
                      }
                      light(bright[0], bright[1], bright[2]);
                      LEDstate();
                  }
                  break;
      case 0x03:  //parameter();
                  break;
      default: break;
    }
  }
}

void light (uint8_t R, uint8_t G, uint8_t B){
  R=R^0xFF;
  G=G^0xFF;
  B=B^0xFF;
  analogWrite(RED, R);
  analogWrite(GREEN, G);
  analogWrite(BLUE, B);
}

void LEDstate(){
  uint8_t led[17];
  led[0]=0x01;
  led[1]=actuatorLED;
  uint8_t pos=2;
  ftohex(float(bright[0]), value);
  for(uint8_t i=0; i<sizeof(value); i++)led[pos+i]=value[i];
  pos+=5;
  ftohex(float(bright[1]), value);
  for(uint8_t i=0; i<sizeof(value); i++)led[pos+i]=value[i];
  pos+=5;
```

```
  ftohex(float(bright[2]), value);
  for(uint8_t i=0; i<sizeof(value); i++)led[pos+i]=value[i];
  tx_request(led, sizeof(led));
 }
```

Resultados

Luego de enviar varios comandos de actuación, a continuación puede verse el aspecto que presentaba el programa. Préstese atención a la columan last de la lista de parámetros y a que, aunque un parámetro no haya sido modificado, al realizar una actuación se envían siempre el estado de los tres parámetros.

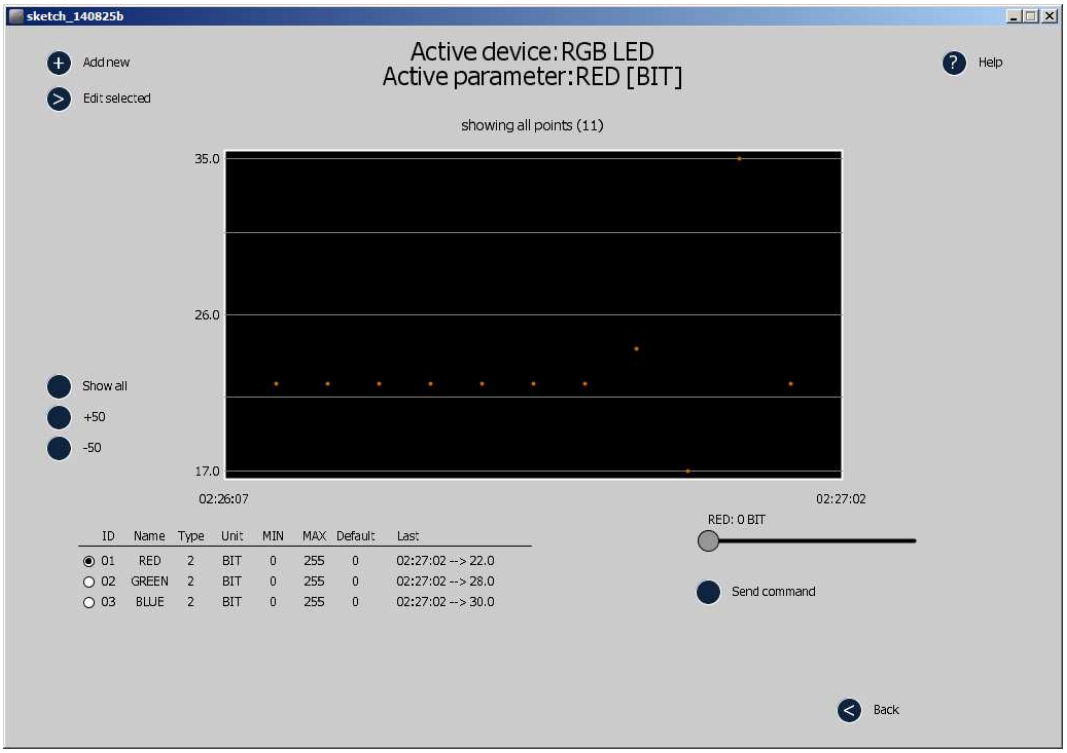

Figura 43.Programa funcionando en el nivel del actuador durante la prueba.

También se muestra el aspecto de diodo led en el que, a pesar de la mezcla, todavía pueden apreciarse los colores individuales.

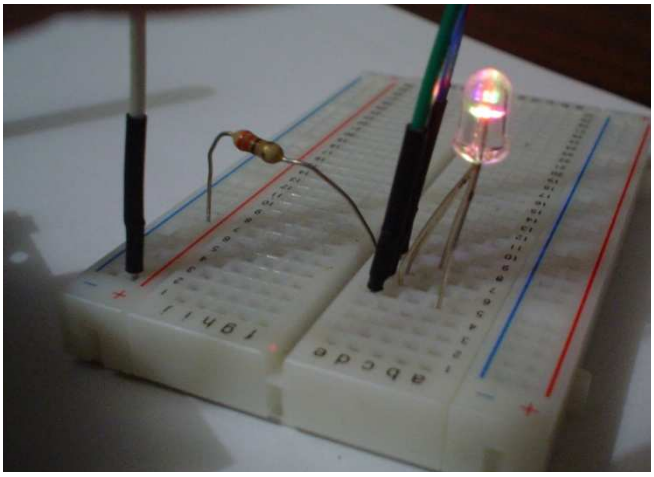

Figura 44. Aspecto del diodo led RGB durante la prueba.

Análisis de los resultados

Tras numerosos comandos se pudo comprobar el perfecto funcionamiento del programa de procesado de datos en el modo de actuación. Por su parte, el programa de Arduino tampoco necesitó de correcciones y su funcionamiento fue, en general, de acuerdo a lo esperado. El resultado de esta prueba fue totalmente satisfactorio.

6.4. Prueba de alcance

Objetivo

El objetivo de esta prueba es estudiar el comportamiento de la red de comunicaciones en una estructura multi-hop.

Material utilizado

Para la realización de esta prueba se utilizaron tres nodos para formar la red: el nodo central, un nodo router y un nodo sensor. El primero fue conectado al ordenador para recoger los datos; el router estaba conectado mediante el adaptador de corriente; y el nodo sensor funcionaba alimentado con pilas y con el sensor DHT-11 para transmitir datos.

- 3x placas Arduinoo, una sin el MCU.
- 3x Wireless SD shields.
- 3x Radios XBee.
- 1x Sensor DHT-11
- 1x Pack de pilas recargables AA
- 1x Cable USB
- 1x Adaptador AC/DC
- Cables de conexión

Metodología

Con los tres nodos conectados a la red, la radio router se configuró con el modo SLEEP deshabilitado; mientras que la radio del nodo sensor se configuró con el modo SLEEP habilitado y periodos de 30 segundos apagado y 5 encendido. El resto de los parámetros se configuró igual que en las pruebas anteriores.

Tabla 21. Resumen de la configuración de los parámetros de las radios.

Parámetro	Nodo central	Nodo router	Nodo sensor
Perfil	Coordinador API	Router API	End device API
ID	2014	2014	2014
AP	1	1	1
DH	00 13 A2 00	00 13 A2 00	00 13 A2 00

DL	40 BA BC 33	40 BA BC 78	40 BA BC 33
SM	-	00	4
SN	-	-	03
SP	-	-	03 E8
ST	-	-	13 88
SO	-	-	2

La prueba se llevó a cabo utilizando el software X-CTU para estudiar el comportamiento de la red. El procedimiento experimental consistió en separar progresivamente los nodos observando cómo se reorganizaba la red y cuándo se recibían los datos desde el nodo sensor. Para ello se trabajó en X-CTU en el modo de consola para estudiar los mensajes recibidos y en el modo de red para estudiar la disposición de la red. En el modo de red se hizo uso de la aplicación de escaneo de la red, que permite descubrir todas las radios XBee en modo API que se encuentren al alcance de cualquier nodo de la red, ya sea que estén o no conectadas a ésta.

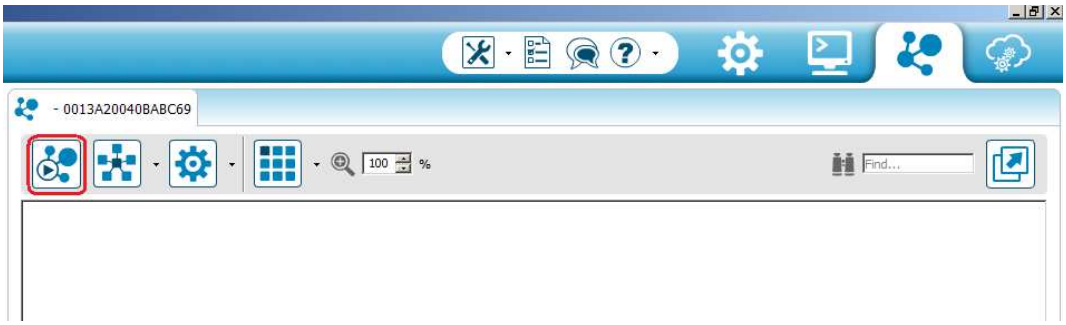

Figura 45. Detalle de X-CTU funcionando en el modo de red. Señalado en rojo: botón para iniciar la aplicación de escaneo de la red.

Resultados

Inicialmente los tres nodos estaban separados una corta distancia dentro de la misma habitación. Luego el nodo coordinador, conectado a un ordenador portátil, se fue alejando por la calle del resto, situando edificios y otros obstáculos del entorno urbano en la línea de visión. Cabe destacar que la separación se incrementó no sólo en horizontal sino también en vertical. Por su parte, la separación entre el router, situado ahora al aire libre para mejorar la recepción con el coordinador, y el nodo sensor se mantuvo constante a menos de cuatro metros. El alcance máximo obtenido durante la prueba fue de unos 58 metros.

Figura 46. Localización de la prueba y situación de los nodos. Feunte Google Maps

Uno de los aspectos más interesantes de esta prueba fue estudiar la organización de la red. A pesar de que se especificó en la configuración de las radios que la End device debía conectarse al router y éste al coordinador, la función de escaneo de la red de X-CTU mostraba que ambas radios se encontraban conectadas al nodo coordinador de forma independiente, formando una red en estrella, como puede verse en siguiente imagen del programa funcionando en el modo de red.

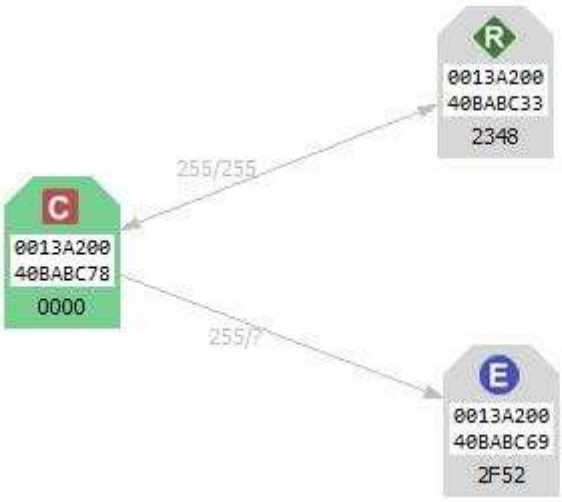

Figura 47. Disposición de la red al momento de iniciar la prueba.

Tras incrementar la distancia de separación lo suficiente y tras perder momentáneamente la recepción de los datos del sensor, se ejecutó la aplicación de escaneo de la red que mostró la siguiente organización de las conexiones que conformaban una red multi-hop:

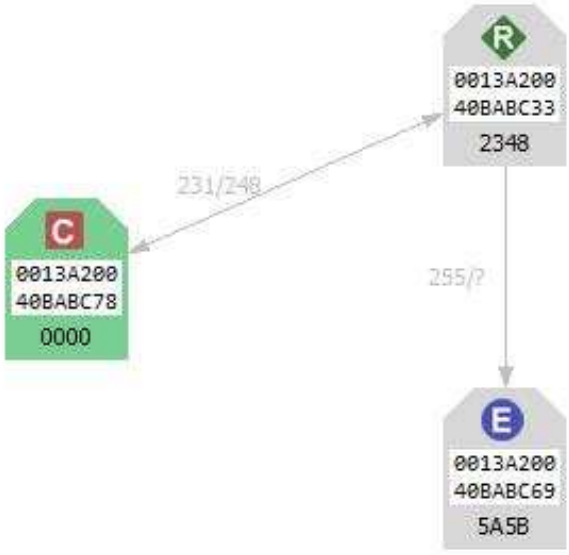

Figura 48. Disposición de la red al momento del máximo alcance.

En esta prueba también se evaluó el tiempo que se tardaba en recibir los datos a través de la red. El ciclo de SLEEP/WAKE UP configurado en la radio end device del nodo sensor dura unos 35 segundos (30 segundos apagado y 5 segundos encendido). Teniendo en cuenta esto se tomaron muestras de la frecuencia con que se recibían los mensajes del nodo sensor desde la consola de X-CTU al iniciar la prueba, es decir, con los nodos juntos; y al momento de máximo alcance, es decir, cuando los nodos estaban dispuestos según la figura anterior.

En la siguiente figura puede verse el registro de los mensajes recibidos en la consola al iniciar la prueba, con la hora respectiva en la que se recibieron, y a continuación puede verse el resumen de ambos casos con la hora en que se recibieron los mensajes y la frecuencia resultante.

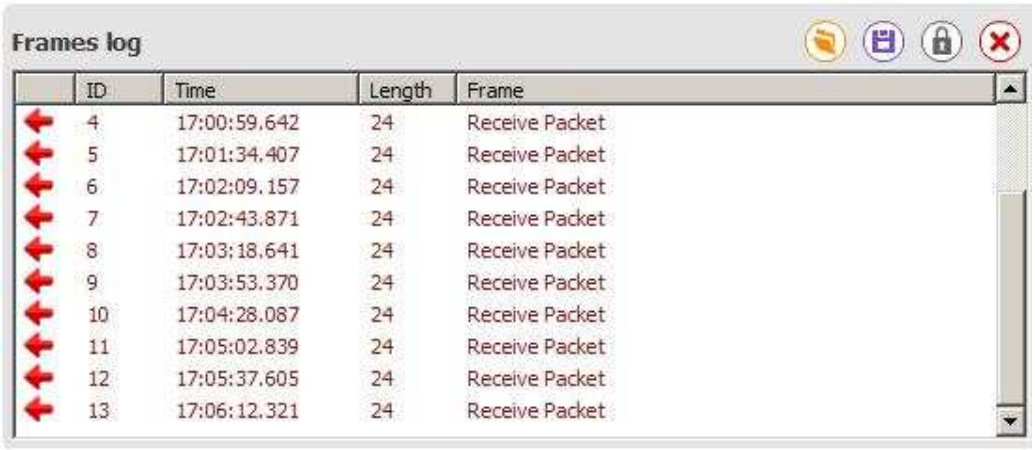

Figura 49. Registro de mansajes recibidos con los nodos en estrella.

Tabla 22. Resultados de la frecuencia de los mensajes captados.

Multi-hop		**Estrella**	
t	**Δt**	**t**	**Δt**
39:56,0		01:34,4	
40:32,2	00:36,2	02:09,2	00:34,8
41:08,5	00:36,3	02:44,0	00:34,8
41:44,7	00:36,2	03:18,6	00:34,6
42:21,0	00:36,2	03:53,4	00:34,7
42:57,3	00:36,3	04:28,1	00:34,7
		05:02,8	00:34,8
		05:37,6	00:34,8
		06:12,3	00:34,7

Análisis de los resultados

Esta prueba resultó muy útil por varias razones, la primera de las cuales era realizar un ensayo práctico de alcance de los XBee a plena potencia en un entorno urbano. El alcance conseguido de 58 m resultó muy satisfactorio y próximo a lo esperado de acuerdo con las características del fabricante, que indican un alcance máximo de 60 m en entorno urbano.

En segundo lugar esta prueba fue una demostración de las capacidades del protocolo ZigBee. La capacidad de autoformación de la red y el llamado self-healing o capacidad de restauración se mostró en las dos etapas claves del

ensayo. Por una parte al iniciar la prueba y dado que los nodos estaban todos al alcance de coordinador, la red se configuró automáticamente en estrella para acortar la ruta de los mensajes. Por otro lado, cuando el end device estuvo fuera del alcance del coordinado, la red se reorganizó pasando los mensajes a través del router para que estos pudieran llegar finalmente al coordinador. Ambas situaciones quedaron reflejadas en el escaneo de la red realizado en X-CTU y las imágenes que aquí se muestran.

Esta capacidad es interesante para un posible estudio futuro de la formación y administración de redes con dispositivos móviles bajo protocolo ZigBee.

El tiempo y los retrasos en la red resultan un poco más difíciles de explicar. Por un lado puede verse que la frecuencia de datos con la disposición en estrella es muy próxima a la frecuencia del ciclo del módulo XBee. Incluso se ve que es ligeramente inferior a los 35 segundos, aunque no se ha podido determinar la razón de esta desviación, probablemente sea un error en la configuración de los parámetros de la radio.

Por su parte, cuando la red se encontraba en la disposición multi-hop puede verse que el periodo entre mensajes se ha incrementado en 1,5 segundos, aproximadamente respecto de la otra disposición. Este significativo incremento puede deberse a la forma en que se transmiten los mensajes en modo API.

De acuerdo a la bibliografía consultada [5] [7] cuando una radio en modo API tiene que transmitir un mensaje, primero envía mensajes de direccionamiento para tratar de encontrar el destinatario, de acuerdo a la dirección indicada, y determinar la ruta que se utilizará. A continuación se envía el mensaje y, si así se ha indicado en el marco API de transmisión, el destinatario responderá con un mensaje de acknowledgment (acuse de recibo).

Estas tareas requieren tiempo y mantienen al módulo XBee encendido durante un periodo más largo del indicado. Aunque se creía que el ciclo de encendido/apagado de XBee sería de 35 segundos de acuerdo al parámetro SP y ST, la realidad es que su comportamiento no es exacto. El parámetro SP indica el tiempo que el módulo Xbee permanecerá apagado, el cual se puede extender con el parámetro SN. El parámetro ST no indica exactamente el tiempo que la radio permanecerá encendida sino que indica un periodo de espera o ,como su nombre indica, un time before sleep. Cuando la radio pasa un periodo de tiempo ST sin recibir datos desde la red o el UART entrará en modo SLEEP y se apagará iniciando el temporizador SP. Pero si mientras la radio se encuentra encendida se

reciben datos, el temporizador ST se reinicia y esperará a que haya silencio durante el tiempo especificado para apagar el dispositivo.

Como resultado, el tener que realizar el proceso de direccionamiento, el recibir el acuse de recibo del destinatario; y el propio sensor utilizado que parece ser lento ya que puede necesitar hasta 250 ms para transmitir una medición a Arduino [9], alargan el periodo que la radio permanece encendida y, en consecuencia, el tiempo total del ciclo.

Aunque en general el resultado de la prueba fue satisfactorio, sobre el tema de los retrasos serían necesarios más ensayos para evaluar en qué medida se incrementa el retraso en la recepción de los mensajes, cuando el número de saltos que deben realizar aumenta. También sería útil estudiar cómo influye el tráfico de la red en el retraso.

Un último apunte sobre este tema está relacionado con los parámetros del SLEEP mode del nodo coordinador. Los nodos con este perfil no pueden entrar en modo SLEEP y apagarse, ya que al ser el que administra la red debe permanecer encendido. A pesar de ello, hay dos parámetros del SLEEP mode de un coordinador que se pueden modificar: SP y SN. La configuración de estos parámetros indicará durante cuánto tiempo el nodo coordinador almacenará un mensaje destinado a un nodo que no responde (porque está apagado en modo SLEEP, por ejemplo). Pasado este tiempo y si no se pudo enviar, el mensaje se descarta para liberar memoria. Pero además, y como se indica en X-CTU, si un nodo no responde al coordinador durante el periodo de tiempo especificado por estos parámetros, será eliminado de la tabla de dispositivos quedando fuera de la red. Como resultado, cada vez que el nodo se encienda deberá pasar por el proceso de volver a incorporarse a la red y esperar a que el coordinador le asigne una dirección (dirección de 16 bits) y busque una ruta para enviar sus mensajes.

Si no se configuran estos parámetros de acuerdo a los nodos de la red, es posbile que el retraso en las comunicaciones se incremente

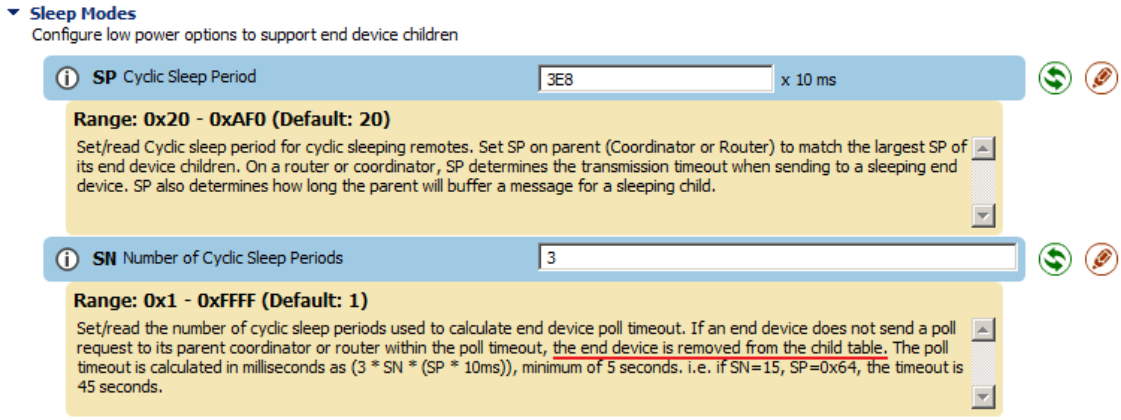

Figura 50. Explicación del significado de los parámetros del Sleep mode para un nodo coordinador en X-CTU.

6.5. Prueba consumo energía

Objetivo

El objetivo de esta prueba fue estudiar el consumo de energía de los nodos de la red.

Material utilizado

Para la realización de esta prueba se utilizó el mismo material que en la prueba de larga exposición. Además de esto se utilizó un multímetro digital para medir la corriente consumida por el nodo.

- 2x placas Arduinoo, una sin el MCU.
- 2x Wireless SD shields.
- 2x Radios XBee.
- 1x Sensor DHT-11
- 1x Pack de pilas recargables AA
- 1x Cable USB
- Cables de conexión
- 1x Multímetro digital

Metodología

En esta prueba el nodo sensor estaba configurado con el modo SLEEP habilitado y con el mismo ciclo que en la prueba anterior: 30 segundos apagado y un periodo encendido de 5 segundos. En esta prueba el parámetro de potencia de emisión de la radio se configuró a su nivel por defecto, el máximo, para determinar el consumo máximo de energía.

Tabla 23. Resumen de la configuración de los parámetros de las radios.

Parámetro	Nodo central	Nodo sensor
Perfil	Coordinador API	End device API
ID	2014	2014
AP	1	1
DH	00 13 A2 00	00 13 A2 00
DL	40 BA BC 33	40 BA BC 33

PL	4	4
SM	-	4
SN	-	03
SP	-	03 E8
ST	-	13 88
SO	-	02

Por su parte el nodo coordinador estaba conectado al para recibir las mediciones desde el terminal de X-CTU.

Durante la prueba el nodo sensor enviaba sus mediciones al nodo central mientras era alimentado por las pilas. El multímetro estaba conectado para medir el consumo de corriente del nodo durante la prueba, particularmente la diferencia entre el nodo encendido y apagado.

Resultados

Durante la prueba se pudo medir que el nodo consumía unos 65 mA cuando estaba encendido; mientras que estado SLEEP el nodo tenía un consumo de 24 mA. Con el fin de determinar el consumo exacto del nodo se desconecto el sensor DHT-11 y se midió un consumo de 20 mA en estado apagado y de 62 mA con la radio encendida.

Cabe destacar también que se registró un pico de más de 90 mA al momento de arrancar el nodo y conectarse el XBee a la red.

Análisis de los resultados

Con esta se confirmaron las enormes ventajas de trabajar con el SLEEP mode, en lo que a consumo energético se refiere. Sólo teniendo en cuenta el nodo (Arduino + XBee), el consumo de energía se reduce hasta casi un 68 % al apagar los dispositivos. Al tener en cuenta sensores y otros dispositivos se reduce la cifra, aunque en el caso de esta prueba, con el sensor DHT-11 el consumo se redujo en un 63 %.

Conociendo el consumo del nodo en condiciones de encendido y apagado se puede estimar el consumo promedio con la siguiente fórmula:

$$I = \frac{I_{ON} \cdot t_{ON} + I_{OFF} \cdot t_{OFF}}{t_{ON} + t_{OFF}}$$

Y con este dato se podría estimar la duración de las baterías. Para el caso de esta prueba, teniendo en cuenta el consumo del nodo con el sensor y el paquete de pilas recargables AA conectadas en serie de 2300 mAh, la autonomía sería de:

$$T = \frac{C}{I} = \frac{t_{ON} + t_{OFF}}{I_{ON} \cdot t_{ON} + I_{OFF} \cdot t_{OFF}} \cdot C$$

$$T = \frac{5 + 30}{65 \cdot 5 + 25 \cdot 30} \cdot 2300 = 74{,}88\ horas$$

Ésta es una primera aproximación y es necesario recordar que el voltaje de una pila no se mantiene constante a medida que se descarga. A modo ilustrativo, al iniciar la prueba de larga exposición (prueba 2) el voltaje del pack de pilas era de más de 7,5 V. Tras las quince horas que duró la prueba se volvió a medir el voltaje, que ahora no llegaba a los 6 V. Por su parte, a medida que el voltaje cae, la corriente que la pila es capaz de suministrar también disminuye. Si las pilas cargadas suministran 20 mA al nodo cuando está apagado, al descargarse la corriente no alcanzaba los 14 mA.

Los nodos sensores se alimentan a través del jack de la placa Arduino, el cual se encuentra conectado a un robusto regulador de voltaje. Este componente requiere una especial atención por el papel que juega en el consumo de energía. De acuerdo con la bibliografía consultada [6] el regulador requiere de un voltaje mínimo para empezar a suministrar a la placa los 5 V con los que funcionan el resto de componentes. El umbral del regulador ronda los 6 V, pro además la misma bibliografía recomienda suministrar más de 7 V si se trabaja con pilas.

En lo que a corriente se refiere, buena parte de la energía se pierde por la acción del regulador. Cuando los módulos XBee se encuentran apagados (en modo de ahorro de energía) su consumo, de acuerdo al manual del fabricante, es de alrededor de 3,5 µA. El consumo del AT mega 328p, el MCU del Arduino también ronda esta magnitud, nunca superando los 10 µA. Por tanto, la corriente que consume el nodo cuando se encuentra apagado, en modo sleep, se debe básicamente al regulador de voltaje. Supone por tanto la principal ineficiencia y es un aspecto que habría que mejorar en futuros desarrollos. A pesar de todo esto, los resultados de esta prueba son muy satisfactorios por el ahorro de energía demostrado al trabajar con el modo sleep.

Cada red creada deberá estar adaptada a las necesidades del usuario/operador. Si los nodos no necesitan de memoria de almacenamiento porque permanecerán conectados siempre a la red, este elemento puede eliminarse, al igual que el RTC. Si hay disponible una fuente de energía cerca, puede eliminarse la batería. Y llegado el caso, las radios XBee pueden ser capaces de administrar entradas y salidas digitales, y disponen de un ADC de 10 bits para mediciones analógicas.

De esta forma el nodo sensor se puede simplificar tanto que sólo se conserve el módulo Xbee y no sea necesario ni siquiera el microcontrolador Arduino. El nuevo XBee ZB *programmable* incluye un segundo procesador programable con una aplicación llamada Code Warrior. Este procesador que implementan las radios XBee no tiene nada que envidiar al ATmega328 utilizado en las placas Arduino, de hecho sus capacidades de almacenamiento son iguales (32 kB de memoria flash y 2 kB de RAM).

7. Impacto ambiental

Tras la explicación del sistema desarrollado, en este capítulo se analizan los aspectos ambientales relacionados con el proyecto y se plantean posibles soluciones para reducir su impacto. El capítulo ha sido dividido en las tres etapas principales del proyecto, se acuerdo a la *Guia de continguts ambientals de projectes* [10] de fi de carrera de la UPC.

7.1. Diseño

En esta etapa se incluyen tanto la concepción del proyecto, como el desarrollo del sistema, las pruebas de funcionamiento y la redacción de los documentos. Esta fase es de vital importancia ya que de las decisiones que aquí se tomen dependerá el impacto ambiental que el sistema tendrá en el futuro.

Varios aspectos del diseño del sistema ayudan a reducir el impacto ambiental del sistema. El uso de componentes modulares permite su reutilización o reemplazo de forma sencilla, reduciendo el volumen de residuos generados. La implementación del modo SLEEP en los módulos ayudará a reducir el consumo de los nodos sensores.

Otro aspecto a considerar es que el desarrollo de una red de comunicaciones mesh con nodos ZigBee de baja potencia, reduce las perturbaciones electromagnéticas en el ambiente en comparación con otras tecnologías como las de telefonía móvil o Wi-fi. Mientras los router WiFi domésticos suelen rondar los 20 dBm (100 mW) [11] de potencia de transmisión, una radio Xbee PRO S2B, como la seleccionada para este proyecto, tiene una potencia de 10 dBm (10 mW) en su versión internacional. Esta diferencia es incluso mayor cuando se compara con la potencia de los routers utilizados en los llamados Hot-Spots públicos que puede llegar a los 700 mW.

En cuanto al desarrollo del proyecto, el impacto estuvo limitado, básicamente, al consumo de electricidad para la redacción de los documentos del proyecto en ordenador. También habría que tener en cuenta el uso de energía de los nodos durante las pruebas. Sobre este tema cabe destacar el uso de pilas recargables, con lo que no se generaron residuos de este tipo.

Aunque una parte importante de los componentes electrónicos utilizados en este proyecto fue adquirida expresamente para la realización de este proyecto, podrán ser reutilizados perfectamente en el futuro. Quizás lo más significativo

sea la necesidad de limpiar soldaduras de estaño realizadas, con los consiguientes residuos de estaño y plomo del hilo de soldadura y fragmentos de cable. A pesar de ello, el volumen de residuos es muy reducido.

7.2. Fabricación

La fabricación de los componentes electrónicos es la etapa con mayor impacto ambiental, dada la naturaleza química de algunos de los materiales utilizados. Aunque la fabricación de los dispositivos electrónicos no es un aspecto tratado específicamente en el proyecto, los efectos ambientales derivados de su producción seguirán estando presentes.

La fabricación de productos electrónicos comercializados en Europa debe realizarse de acuerdo a la normativa legal vigente, la directiva europea 2011/65/CE, también conocida como RoHS (Restruction of Hazardous Substances). Esta directiva restringe el uso de ciertos materiales y sustancias peligrosas en productos eléctricos y electrónicos. La normativa también expone las obligaciones de los importadores y de los estados miembros, quienes velarán porque sólo se introduzcan en el mercado aquellos productos que cumplen la normativa. [12]

De acuerdo con la información de los fabricantes, tanto las placas Arduino como las radios Xbee cumplen con esta normativa. [13]

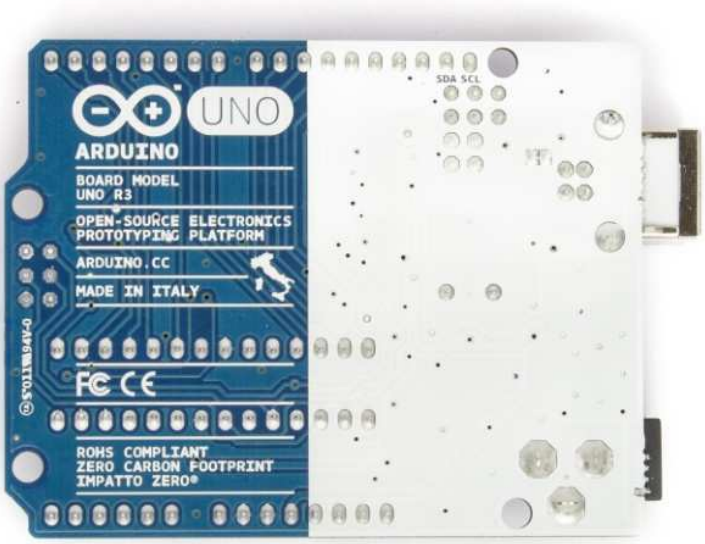

Figura 51. Reverso de la placa Arduino mostrando el cumplimiento de la RoHS. Fuente Arduino.

Por otro lado habrá que considerar el impacto relacionado con la distribución de los componentes hasta el consumidor final. El estudio de este apartado es particularmente difícil porque dependerá de las condiciones en que se realice el transporte: cantidad, tamaño, peso, empaque, medio de transporte y distancia. A pesar de ello, se puede afirmar que los productos populares que se transportan en grandes cantidades y que tienen una alta disponibilidad, minimizan el impacto al mejorar la eficiencia energética del transporte y reducir la necesidad de desplazamiento del usuario.

7.3. Utilización

El impacto ambiental del sistema desarrollado durante la etapa de utilización es bastante reducido y se limita, principalmente, al consumo energético de los nodos. Teniendo en cuenta los resultados de la prueba de consumo energético, este impacto es bastante reducido para un nodo. Aunque si el tamaño de la red es lo suficientemente importante, cabría mirarlo con mayor atención. En todo caso, el uso de fuentes autónomas renovables como paneles solares ayudará a reducir la dependencia de la red eléctrica. Hacer uso de los modos de ahorro de energía ayudará, también, a reducir el consumo de los nodos y alargar la vida de pilas y baterías, ya que se reducirá el número de ciclos de carga y descarga a que se vean sometidos.

A pesar de la gran fiabilidad de los componentes electrónicos, es inevitable que alguno de ellos sufra un desperfecto en algún momento. Aquí cobra importancia el uso de los elementos modulares del diseño. Cuando algunos de los módulos deja de funcionar puede ser reemplazado fácil y rápidamente y ser enviado a reparar, si fuera posible, o descartado definitivamente, rediciendo el volumen de residuos de forma efectiva.

7.4. Desmantelamiento

Cuando el sistema deje de ser utilizado, bien porque la aplicación en la que se utilizaba ya no lo necesita, o bien porque serán reemplazados por otros componentes con mejores características, los módulos pueden reprogramarse y reconfigurarse fácilmente para adaptarlos a futuras aplicaciones. Siempre que los componentes se encuentren en buen estado es recomendable guardarlos para una futura reutilización.

Por otro lado, si los componentes han llegado al final de su vida útil o ha sufrido un desperfecto irreparable, habrá que considerarlos como residuos. Deberán

tratarse con especial atención las pilas o baterías que ya no sean útiles debido a la toxicidad de las sustancias que las componen. El tratamiento de los residuos generados deberá realizarse de acuerdo a la legislación vigente en materia de residuos de elementos electrónicos, que en este caso es la directiva 2012/19/CE. La llamada WEEE (Waste Electrical and electronic Equipments) establece un marco regulatorio para redcir el impacto medioambiental relacionado con la generación y la gestión de residuos de aparatos eléctricos y electrónicos. [12]

8. Normativa

Para la realización de este proyecto no fue necesario evaluar aspectos normativos específicos, pues la gran mayoría de estos es responsabilidad de los fabricantes de los dispositivos electrónicos empleados. Son ellos quienes deben asegurar el cumplimiento de la normativa en las etapas de diseño, manufactura y distribución.

La gestión y el tratamiento de los residuos generados por los aparatos eléctricos y electrónicos utilizados en el sistema desarrollado deberá realizarse por entidades autorizadas, siguiendo la legislación vigente. Es responsabilidad del usuario, en este punto, deponer los residuos generados de forma adecuada para facilitar su gestión.

En aplicaciones concretas deberá prestarse mayor atención a este apartado. Es posible que debido al origen o la naturaleza de los datos recogidos estos deban ser tratados de acuerdo a la legislación vigente en materia de protección de datos personales. En todo caso, será responsabilidad del usuario evaluar estos aspectos.

En todo caso, las principales normas aplicables al sistema desarrollado en este proyecto son:

- Directiva 2004/108/CEE de Compatibilidad Electromagnética
- Directiva 1999/5/CE sobre Equipos de Terminales de Radio y Telecomunicaciones (R&TTE)
- Directiva 2011/65/EU sobre restricciones a la utilización de determinadas sustancias peligrosas en aparatos eléctricos y electrónicos (RoHS).
- Directiva 2012/19/EU Sobre residuos de aparatos eléctricos y electrónicos (WEEE).

En cuanto al apartado técnico, el fabricante de las radios (Digi International) deberá asegurarse de que sus productos se desarrollan no sólo de acuerdo al estándar IEEE 802.15.4, sino también siguiendo las especificaciones técnicas desarrolladas por la ZigBee Alliance.

9.Conclusiones

Todo el trabajo realizado durante los últimos meses, y que ha sido aquí documentado, ha servido para confirmar la viabilidad técnica del principal objetivo del proyecto. Se ha conseguido construir un nodo sensor que comunica los datos recogidos a través de una red de comunicaciones bajo protocolo ZigBee. La formación y configuración de la red se ha explicado con la ayuda del programa utilizado para esta tarea: X-CTU. El programa de procesado de datos desarrollado en Processing ha mostrado excelentes resultados en las pruebas de funcionamiento llevadas a cabo. Finalmente, las pruebas realizadas han confirmado el correcto funcionamiento del sistema diseñado.

En cuanto a las especificaciones del proyecto, la única que no ha podido cumplirse fue la del coste del nodo sensor. El empleo de radios más potentes pero también más caras y la necesidad de incorporar un adaptador para comunicarse con Arduino (la SiWireless SD shield), incrementó el coste del nodo. Esto puede verse en detalle en el documento de presupuesto. A pesar de ello, la posibilidad de realizar pruebas concretas con las tarjetas SD y con las radios se considera que justifican el gasto adicional. En todo caso, si se desea implementar el sistema aquí desarrollado, el usuario puede optar por un adaptador más sencillo o por radios XBee de menor potencia de transmisión y más baratas. La decisión final, como se aclaró durante el proyecto, queda en manos del usuario.

9.1. Evaluación de conocimientos

El empleo de las placas Arduino permitió mejorar los conocimientos existentes sobre este dispositivo, adquiriendo nueva experiencia sobre sus capacidades, como el almacenamiento de datos, los puertos de comunicación serie, I2C y SPI o la configuración del MCU para habilitar el modo SLEEP. Por otro lado, y aunque no se emplearon finalmente en el proyecto, también se adquirieron conocimientos sobre el uso de relojes RTC y tarjetas de memoria SD con Arduino.

Los módulos XBee, otra de las especificaciones del proyecto, fueron una novedad ya que no se disponían de conocimientos previos sobre ellos. La disponibilidad de información clara y fiable, tanto en el manual de XBee como, sobre todo, en el libro *Building Wireless Sensor Networks,* principal referencia utilizada para este dispositivo, fueron cruciales para reducir el tiempo de aprendizaje. Con estas referencias se pudo aprender no sólo las bases del funcionamiento de los módulos XBee, sino también los complicados marcos API, aspecto crucial de este proyecto y necesario para la formación de redes multi-hop. Apuntar finalmente

que tras la realización de los ensayos ha sorprendido el enorme potencial de los módulos XBee. El alcance conseguido, la fiabilidad de las comunicaciones y, primeramente, las propiedades del protocolo ZigBee para la autoformación, enrutado y reconfiguración de la red son los aspectos a destacar.

Processing es otro de los puntales de este proyecto y otro de los temas sobre el que no se disponía de experiencia previa. La aplicación desarrollada se programó en base a los conocimientos de otros lenguajes, como C++ o Arduino, y con la ayuda invaluable de la guía de referencias incluida en el programa, los archivos de ayuda. No se consultaron páginas especializadas, ni foros o blogs, sólo dos referencias adicionales sirvieron para solucionar dudas sobre los aspectos gráficos de la aplicación. Tras casi ochenta horas de trabajo Processing ha demostrado ser una herramienta fácil de aprender y muy versátil de aplicar. La posibilidad de desarrollar aplicaciones con gráficos interactivos le otorga una ventaja frente a otros lenguajes, que se aprovechó aquí para crear la aplicación de procesado de datos con una interfaz lo más sencilla posible para el usuario.

9.2. Limitaciones

A pesar de todo el trabajo realizado, se pueden identificar algunas limitaciones en el sistema desarrollado que son susceptibles de mejoras en el futuro. La primera es la construcción de los nodos sensores, que por el uso de módulos apilables resulta en un dispositivo de tamaño considerable, teniendo en cuenta, además, que no hay conectados ningún sensor o circuito de condicionamiento de señal.

Además, por las características eléctricas de la placa Arduino, pero básicamente por la radio, resulta necesario el uso de un paquete de baterías de un tamaño similar al nodo y con un peso bastante alto. Y aunque la implementación del modo SLEEP permite un ahorro importante energía, cabe destacar, como se señalo en las pruebas, que una parte importante de la energía se desperdicia en la tarea de regulación en la placa Arduino.

La interacción entre Arduino e XBee también es una limitación del sistema ya que ha sido necesario programarla expresamente, con la consiguiente reducción de espacio en la memoria del micro controlador. Además, cualquier característica adicional de las radios XBee que se quiera aprovechar en el futuro necesitará ser programada desde cero, pues este proyecto se ha centrado en las funciones básicas de enviar y recibir mensajes desde y hacia la radio en modo API.

En cuanto a la aplicación desarrollada en Processing, una de sus limitaciones está relacionada con los aspectos de automatización y control. Por un lado el programa se centra en la red de comunicaciones, mientras que una aplicación de automatismos estaría centrada en los aparatos, dispositivos y máquinas de la instalación. Por el otro, el control de comandos programado, consistente en un regulador de nivel y un botón para enviar comandos puede resultar demasiado simple para algunas aplicaciones.

Otra limitación se refiere a los gráficos de mediciones del nivel de dispositivo. Actualmente los puntos se grafican a espacios regulares según el número que haya en pantalla y no según la hora en que se registraron, ya que se considera que los sensores siempre envían mediciones a intervalos constantes. Por un lado esto simplificó la tarea de la programación, pero por el otro la aplicación ignora el espacio temporal de las mediciones.

Pero sin duda la principal limitación del sistema desarrollado es la imposibilidad de procesar datos provenientes de la red y configurar las radios de forma remota al mismo tiempo. Estas tareas se realizan desde dos aplicaciones diferentes, por un lado la aplicación desarrollada en Processing, y por el otro X-CTU. Sólo una de ellas puede establecer conexión con el nodo coordinador a la vez mientras que la otra quedará bloqueada. Ahora mismo la única opción posible es cerrar una de las aplicaciones para utilizar la otra, aunque esto puede implicar una pérdida inadmisible de datos.

9.3. Mejoras

Teniendo en cuenta las limitaciones encontradas, aquí se plantean algunas posibles mejoras que puedan servir de base para futuros desarrollos del sistema.

Por un lado se podría otorgar más control de la red a la aplicación de Processing. Se podrían programar las funciones necesarias para la construcción, el procesado y el envío de marcos API de configuración AT remota. Esto eliminaría la necesidad de tener que trabajar con X-CTU para esta tarea. Otra opción más sencilla sería utilizar un nodo específicamente como punto de acceso para realizar las configuraciones, mientras que el nodo coordinador seguiría encargándose sólo de recoger los datos. Aunque más sencilla, los cambios que se realicen de esta forma no quedarán registrados en la aplicación y será tarea del operador hacer que consten.

Otra mejora pasaría por mejorar la interpretación del gráfico de puntos, pudiendo ajustar los puntos mostrados en pantalla a un intervalo de tiempo

configurable. Esto requiere que el programa pueda interpretar el formato de hora con que se guardaron los datos. Además se podrían incluir funciones adicionales para exportar los gráficos, imprimirlos, o realizar curvas de ajuste.

En el apartado de comunicaciones las mejoras que se plantea implementar el direccionamiento de mensajes mediante la dirección de 16 bits. Esto permitirá acelerar la transmisión de los mensajes, reducir el tráfico de la red al mejorar la velocidad, y disminuir el consumo de energía, al acelerar el direccionamiento de mensajes.

Otra mejora en las comunicaciones sería la posibilidad cargar datos desde la nube o enviarlos para almacenarlos aquí. De esta forma se mejoraría la disponibilidad de los datos, que serían accesibles desde cualquier parte del mundo.

La última gran mejora que se plantea tiene que ver con el hardware de los nodos. Desde hace poco Digi ofrece un nuevo modelo de radio XBee disponible en protocolo ZigBee. Se trata del XBee Programmable, un módulo que incluye todas las prestaciones de un módulo XBee, más un segundo procesador programable para que los desarrolladores puedan crear su propio código adaptado a sus necesidades, como en Arduino. Todo esto en el mismo tamaño que un XBee convencional.

Aunque no se ha estudiado en detalle, las especificaciones indican que el segundo procesador dispone de 32 kB de memoria programable y 2 kB de RAM, lo mismo que el AT Mega 328 P del Arduino UNO. El procesador se programa mediante una aplicación disponible en la página del fabricante, Freescale, llamada Code Warrior. Además existe la posibilidad de realizar lo que se conoce como programación "over the air", que significa programar el procesador a través de la red. De esta forma no hace falta ir hasta donde está instalado el nodo para retirarlo y reprogramarlo conectándolo a un ordenador, como Arduino, sino que el programa se puede enviar a través de la red XBee hasta el nodo donde se instalará de forma automática.

A pesar de que el Xbee Programmable no tenga la misma potencia, que Arduino ni su popularidad, se considera que es un dispositivo muy interesante a estudiar. Imagínese el lector un nodo sensor cuyo tamaño sea tan pequeño como un módulo XBee. La construcción sería más sencilla, al eliminar la placa arduino y la Shield de conexión. La interacción entre los procesadores de aplicación y de comunicación es más sencilla ya que el módulo ha sido diseñado con este fin. La posibilidad de programar el nodo "over the air" es una ventaja muy importante.

Sin duda se reduciría el consumo energético del nodo, y el coste total sería inferior al actual: estas radios se pueden adquirir por menos de 50€.

9.4. Conclusiones finales

Este proyecto ha sido muy interesante y estimulante en lo referido a los sistemas de comunicación inalámbrica. No sólo se han podido adquirir nuevos conocimientos para el futuro, sino que también se han ampliado otros en materia de electrónica y programación. El resultado de todo el tiempo dedicado es un sistema que tiene mucho potencial para futuros desarrollos.

El autor del proyecto confía en que éste sirva de referencia para futuros trabajos y espera que despierte el interés de otros proyectistas sobre el tema.

Bibliografía

[1] E. C. Whitman, «SOSUS. The "Secret Weapon" of Undersea Surveillance,» *Undersea Warfare,* Winter 2005, Vol. 7, No. 2.

[2] Libelium Comunicaciones Distribuidas S.L., «Casos de éxito,» [En línea]. Available: http://www.libelium.com/es/case-studies/. [Último acceso: marzo 2014].

[3] F. M. Archundia Papacetzi, *Wireless Personal Area Network (WPAN) & Home Networking,* Cholula, Puebla, 2003.

[4] Microsoft Corporation, «Definición de las siete capas del modelo OSI,» 23 noviembre 2013. [En línea]. Available: http://support.microsoft.com/kb/103884/es. [Último acceso: marzo 2014].

[5] R. Faludi, Building Wireless Sensor Networks, O'Reilly Media, 2010.

[6] Arduino, «Languaje References,» 2014. [En línea]. Available: http://arduino.cc/en/Reference/HomePage. [Último acceso: junio 2014].

[7] Digi International Inc, «XBee ZB User Manual,» 2012. [En línea]. Available: http://ftp1.digi.com/support/documentation/90000976_T.pdf. [Último acceso: junio 2014].

[8] D. Morrissey, «Arduino, Zigbee and Embedded Development,» 25 abril 2010. [En línea]. Available: http://donalmorrissey.blogspot.com.es/2010/04/putting-arduino-diecimila-to-sleep-part.html. [Último acceso: 17 julio 2014].

[9] A. Allan y K. Bradford, Distributed network data, O'Reilly Media, Inc, 2013.

[10] ETSEIAT, «Guia de Continguts ambientals del projectes de fi de carrera,» 2000. [En línea]. Available: http://www.etseiat.upc.edu/curs-actual/Projecte-Final-Carrera/normatives-pfc. [Último acceso: 8 septiembre 2014].

[11] P. B. Espejo, «La tecnología Wi-fi y sus riesgos,» junio 2008. [En línea]. Available: http://www.ecologistasenaccion.es/article11597.html. [Último

acceso: 8 septiembre 2014].

[12] Comisión Europea, «Waste Electrical & Electronic Equipment (WEEE),» 22 08 2014. [En línea]. Available: http://ec.europa.eu/environment/waste/weee/index_en.htm. [Último acceso: 8 septiembre 2014].

[13] Digi International Inc, «RoHS Certificate of Compliance,» 1 junio 2013. [En línea]. Available: http://www.digi.com/support/kbase/kbaseresultdetl?id=2201. [Último acceso: 8 septiembre 2014].

[14] S. K. M. Raghavendra C.S, Wireless Sensor Networks, Springer Science+Business Media Inc., 2004.

[15] L. Kang, «A Smart Transducer Interface Standard for Sensors and Actuators,» de *The Industrial Information Technology Handbook*, Boca Raton, Florida: CRC Press, 2004.

[16] Silicon Laboratories Inc., «The Evolution of,» 2013. [En línea]. Available: http://www.silabs.com/Support%20Documents/TechnicalDocs/evolution-of-wireless-sensor-networks.pdf. [Último acceso: febrero 2014].

[17] D. Gascón, «802.15.4 vs ZigBee,» 17 noviembre 2008. [En línea]. Available: http://sensor-networks.org/index.php?page=0823123150. [Último acceso: 25 febrero 2014].

[18] Digi International Inc., «Wireless Mesh Networking. ZigBee vs. DigiMesh,» 2008. [En línea]. Available: http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf. [Último acceso: 5 marzo 2014].

[19] F. L. Lewis, «Wireless Sensor Networks».

[20] Digi International Inc, «X-CTU Software,» 2014. [En línea]. Available: http://www.digi.com/support/productdetail?pid=3352. [Último acceso: junio 2014].

[21] M. Margolis, Arduino Cookbook, O'Reilly Media, Inc, 2011.

[22] Processing, «References,» 2014. [En línea]. Available: http://www.processing.org/reference/. [Último acceso: junio 2014].

[23] C. Reas y B. Fry, Getting Started with Processing, O'Reilly Media, Inc., 2010.